The Dissertation Committee for Heather Lynn Hanson
certifies that this is the approved version of the following dissertation:

Coordinated Power, Energy, and Temperature Management

Committee:

Stephen W. Keckler, Supervisor

Douglas C. Burger

Lizy K. John

Charles Lefurgy

David Z. Pan

Coordinated Power, Energy, and Temperature Management

by

Heather Lynn Hanson, B.S.; B.A.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2007

Acknowledgments

I would like to acknowledge team members' technical contributions that are included in this dissertation. In addition to the many fruitful brainstorming sessions, the following people have made specific contributions to experimental infrastructure and publications that I have incorporated in my research and this dissertation.

- Karthick Natarajan and I co-developed the simulator for an Alpha pipeline power study we published in the ISLPED conference in 2003. He was responsible for the speculation analysis, while I performed the over-provisioning analysis.

- Charles Lefurgy and Mike Kistler and I collaborated on a TRIPS Sensor Network document that formed the foundation of the coordinated manager ideas.

- Juan Rubio and Karthick Rajamani developed the Pentium M performance, temperature, and power-monitoring infrastructure, including the `log` program that I used as the base for developing the PET manager prototype software. Chandler McDowell developed the LabView program that acquires and processes power probe measurements.

- Karthick Rajamani developed the memory-sensitive microbenchmark suite, `MS-Loops` that we used for performance, power, and thermal characterization on the Pentium M platform.

- Soraya Ghiasi analyzed the experimental error in the Pentium M power measurement system and researched conditional-preference nets.

- Karthick Rajamani, Juan Rubio, Soraya Ghiasi, and Freeman Rawson and I collaborated on several projects and conference papers, including the Pentium M characterization studies and prototype evaluations.

I would like to thank my faculty advisor, Stephen W. Keckler, and co-advisor Doug Burger for welcoming me into the Computer Architecture and Technology (CART) research group. To the CART group: thank you for the camaraderie, research collaboration, reading (and re-reading) my dissertation chapters, and for teaching me how to play cricket.

I appreciate the thoughtful suggestions of my committee members, Stephen W. Keckler, Doug Burger, Margarida Jacome, Lizy John, Charles Lefurgy, and David Pan. I am grateful for the opportunity to work with IBM's Austin Research Laboratory, who loaned both the Pentium M infrastructure and expertise, and would also like to thank the members of the Power Hour reading discussion group for thought-provoking discussions.

Finally, I cannot thank my family enough for their love and support.

Coordinated Power, Energy, and Temperature Management

Publication No. _____

Heather Lynn Hanson, Ph.D.
The University of Texas at Austin, 2007

Supervisor: Stephen W. Keckler

Power and thermal effects have emerged as serious problems for computing systems by limiting performance, degrading reliability, and imposing a high cost in energy resources. A fundamental problem with power and thermal management is the difficulty of reducing power and heat output without sacrificing performance, which creates a complex web of inter-related constraints and requirements. Meeting these multiple, potentially conflicting objectives simultaneously is a difficult challenge, exacerbated by shifting environmental conditions and variable workloads, yet essential for future generations of high-performance systems.

We propose a comprehensive, goal-oriented management framework that sorts priorities and balances conflicting goals, named **PET** for performance, power, energy, and temperature. The approach provides a level of indirection between macro objectives, such as reducing operating cost or increasing performance, and micro directives, including voltage and frequency settings and other power-management

choices. Goal-driven decisions reflect relevant run-time conditions, rather than predefined policies, and a concise specification for desired outcome provides an opportunity to customize operation to conserve or spend power resources as situations warrant, delivering performance on demand.

We demonstrate the feasibility and benefit of the PET approach with a prototype implementation developed in software, executing on an instrumented Pentium M system. First, we present a detailed characterization of the system response to power management mechanisms to identify the timescales and magnitude of expected response to management decisions. Second, we illustrate PET operation with realistic workloads and usage scenarios, demonstrating that the prototype achieves the desired ranges of operation with dynamic run-time control.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Power and thermal effects have emerged as serious problems for computing systems by limiting performance, degrading reliability, and imposing a high cost in energy resources. A fundamental problem with power and thermal management is the difficulty of reducing power and heat output without sacrificing performance. Only in rare circumstances, such as suppressing spurious time-consuming and power-wasting activities, does a maneuver enhance performance while reducing power consumption and heat generation. In the typical case, performance improvements are limited by power and thermal budgets; power-thermal optimizations are constrained by performance expectations. Performance-enhancing mechanisms, including faster clock speeds, larger caches, sophisticated predictors, complex pipelines, adding more cores, or increasing communication bandwidth, generally consume more power.

Conversely, power and thermal management techniques such as limiting activity on the chip, reducing voltage and frequency, powering down components, or migrating threads each impose a performance penalty or overhead. A performance-aware power and thermal management scheme must balance the opposing forces of enabling performance within limited power and thermal envelopes.

Compounding the problem of balancing performance with power, energy, and temperature constraints is the lack of consensus on a balancing point. A mission-

critical application may warrant maximum possible performance yet a low-priority background task may be served with more modest performance at substantial power savings. From moment to moment, workload demands fluctuate, shifting the power-performance balance point to meet response time requirements for quality-of-service contracts. Mobile systems compromise on performance for energy efficiency while powered by limited battery energy and switch to a high-performance mode when plugged into a power outlet. A single power or thermal management policy will not suit every situation. A management scheme must be flexible to accommodate multiple requirements and constraints that vary through time.

In this dissertation, we examine power consumption and heat generation as computing resources, identifying the components of a microprocessor power budget and gauging responses to power, energy, and thermal management. We describe a comprehensive, goal-oriented management framework that sorts priorities and balances conflicting constraints, named **PET** for coordinating **Performance, Power, Energy, and Temperature** management. The approach provides a level of indirection between macro objectives, such as reducing operating cost or increasing throughput, and micro directives, including power-management and low-level actuator settings. Goal-driven decisions reflect relevant run-time conditions, rather than pre-defined policies, and a concise specification for desired outcome provides an opportunity to customize operation to conserve or spend power resources as situations warrant, delivering performance on demand.

In this chapter, we briefly describe the challenges for effective management and outline our approach to address these challenges. We conclude the chapter with

a thesis statement and summary of contributions.

1.1 Challenges

Performance-aware power and thermal management face several challenges, from process variation in semiconductor fabrication through unpredictable software behavior.

1.1.1 Technology Trends

The observation known as Moore's Law states that the complexity of integrated circuits for the minimum manufacturing cost doubles approximately every two years [64]. The same scaling trends that have enabled generations of ever-increasing performance by providing more transistors per die and enabling faster clock rates have also created a power liability by creating small transistors that switch quickly and leak current even while nominally inactive. A transistor's dynamic power depends upon the capacitance, voltage, and switching rate. Although capacitance per transistor and voltage supply typically scale down with successive generations of fabrication technology, an increase in switching rate and transistor density causes the overall dynamic power to increase. Subthreshold leakage and gate oxide tunneling create currents through nominally "off" devices, causing static power dissipation. Subthreshold leakage is exponentially dependent on temperature; heat dissipated from a high-power device causes even more static power.

Power density has escalated due to increasing power consumption coupled with die sizes essentially unchanged through time. High power density triggers me-

chanical failures and degrades reliability; removing the ever-increasing heat output requires expensive cooling solutions or slowing the frequency gains that enabled performance. Clock rates no longer scale to faster frequencies with each new generation at the same rate as previous generations [3], prompting a renewed interest in parallel computation to increase performance, rather than clock frequency alone. Several processors, including IBM's POWER4 [5] and POWER5, and Intel's CoreDuo [65] have applied the generous transistor count in recent years to create two high-performance cores, including caches and communication support, per die. Commercial product announcements indicate a continuing trend in chip multiprocessors (CMPs) with multiple cores sharing a die in the future: projects include AMD's K8L (also known as Barcelona) and Intel's Kentsfield and Clovertown 4-core processors, Sun Microsystems' 8-core Niagara 2, and IBM's Cell processor with a standard processor core plus 8 "synergistic processing engines." Intel Corporation has also announced a TeraFLOP initiative to create an 80-core processor [66]. However, boosting performance by using more transistors and interconnect, whether by creating large sophisticated cores or by harnessing many smaller cores, still incurs a cost for both dynamic power and static leakage power.

1.1.2 Conflicting Mechanisms

Run-time management decisions must make appropriate choices for a wide range of behavior. Commercial computing systems are typically equipped with dynamic voltage and frequency scaling (DVFS), clock throttling, power-down modes, and other power-management techniques to help manage the growing problems of

power and thermal management. However, managing one aspect does not guarantee a desirable response in others. For example, reducing power by lowering the frequency and voltage may degrade performance and can cause the total energy to increase due to longer execution time; minimizing energy by completing tasks quickly improves performance at the expense of potentially exceeding power limits; low average power can disguise short spikes of high activity that exceed power supply ratings; total power within an acceptable range does not prevent localized thermal hotspots. Propagating workload characteristics and system response to a centralized manager to organize a coherent response in a large system would be impractical. Localized managers with detailed information are better equipped to adapt to variation; however, individual actions may interact in destructive ways.

1.1.3 Variability in Live Systems

Variation inherent throughout the hardware-software stack poses a challenge for effective power and thermal management. Workload variation (software) and component manufacturing variation (hardware) combine to create an environment of uncertain system response to power and temperature management mechanisms.

Software variation has been recorded on many timescales, from microseconds to days. Figure 1.1 shows an example of web server utilization over a 24-hour period for two profiles, the New York Stock Exchange (NYSE) and the 1998 Winter Olympics held in Nagano, Japan [33]. The workload profiles exhibit both high-intensity and low-intensity phases, which translate to corresponding system utilization phases. Provisioning the system for average or typical traffic would sig-

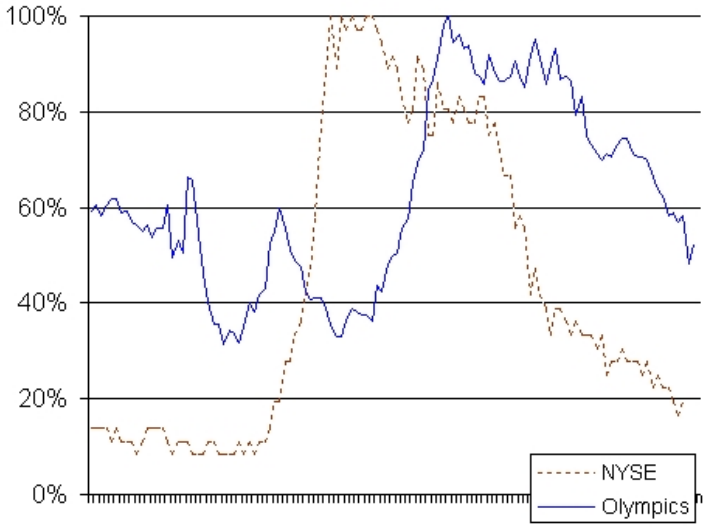


Figure 1.1: Web server utilization patterns from NYSE and Nagano Winter Olympics, over a 24-hour period.

nificantly degrade performance at peak access times; maintaining the server at peak performance levels at all times would waste considerable power during low-activity phases. To achieve throughput to meet workload demands without wasting energy, a power-management scheme must adapt to the changing workload.

In hardware, variability of each chip due to manufacturing process variations leads to a wide range in leakage current and switching speed. Kim, *et al.* note that a ten percent change in transistor gate length can cause a factor of three difference in subthreshold leakage current [53]. Rubio found a ten percent difference in total power consumption among five nominally identical processors [33]. The combined effect of software and hardware variation creates individual, at times unpredictable, system response.

The response to power- and thermal-management techniques varies with workload characteristics. One of the most effective techniques, DVFS, is particularly sensitive to workload memory patterns because altering the clock frequency changes the relative speed between the core processor and off-chip main memory. Compute-bound workloads are more sensitive to frequency change than memory-bound workloads. For example, a frequency change from 2 GHz to 1.2 GHz causes a wide range in performance reduction for SPEC CPU2000 benchmarks executing on a Pentium M platform. The memory-bound `swim` benchmark, for example, tolerates the lower frequency with minimal performance loss, about 1% slowdown, because memory accesses are the performance bottleneck. Changing the CPU frequency does not alter memory access time, and thus lower frequency translates to fewer CPU clock cycles spent waiting for memory accesses. In contrast, workloads that are limited by the rate of computation exhibit more noticeable performance degradation when the core frequency, and thus rate of computation, are reduced. The 40% reduction in frequency lead to a 29% slowdown for `gap` and 40% slowdown for `sixtrack` [72].

Other power and thermal management techniques, such as cache leakage control, would also elicit a range of behavior depending upon cache and memory access patterns. In addition to workload-specific responses, environmental conditions such as air flow also play a role in determining thermal response. Pre-set policies will not always guarantee the desired response, nor could pre-set policies exploit management options to adapt to present workloads and conditions. For example, a single pre-determined frequency would not suit a performance or power target due

to run-time power variation. A conservative frequency would reduce performance of naturally low-power benchmarks and operating conditions while an aggressive high frequency could violate power thresholds for higher power conditions.

1.2 Thesis Statement

The problem of enabling performance with limited power and cooling resources creates a complex web of inter-related constraints and requirements. Meeting these multiple, potentially conflicting objectives simultaneously is a difficult challenge, exacerbated by shifting environmental conditions and variable workloads, yet essential for future generations of high-performance systems. My hypothesis is that defining a framework to express user-defined objectives in terms of multiple conditions simultaneously will enable effective run-time control.

Framework: The approach provides a level of indirection between macro objectives, such as reducing operating cost or delivering high performance, and micro directives, including voltage and frequency settings and other power-management choices. The PET framework principles are briefly described as follows:

1. goal-oriented decisions
2. multi-dimensional objectives
3. orchestrated management mechanisms
4. run-time adaptation.

First, goal-oriented decisions express the desired outcome at a high level, and allow lower levels with access to pertinent information to choose the exact mechanisms to reach the goal. Goal-driven decisions reflect relevant run-time conditions, rather than blindly following pre-defined policies. A concise specification for desired outcome provides an opportunity to customize the goal targets to high performance, low power, or any point in between to conserve or spend power resources as situations warrant, delivering performance on demand.

Second, multi-dimensional objectives express the desired outcome in terms of performance, power, energy, temperature, etc. Rather than condensing the objective into a single metric such as the energy-delay product that obscures useful information about both energy and delay, our multi-dimensional specification accommodates individual metrics for each dimension. A key feature of our approach is defining a relationship between individual metrics with a multi-dimensional space, such as a two-dimensional surface to relate performance and power, or a three-dimensional volume for performance, power and temperature.

Third, computing systems offer several mechanisms for power and thermal control, such as dynamic voltage and frequency scaling, clock throttling, variable fan speed, leakage current reduction techniques, and more. Our framework is designed to orchestrate the response of multiple mechanisms to avoid conflicts from independent opposing actions.

Fourth, PET managers continuously track system behavior and objectives at run-time, tuning settings in response to changing application demands and resource availability. With these principles, The PET framework can adapt to changing work-

loads and environments, pushing the system to the edge of limits while maintaining safe operating conditions [31].

Prototype: We demonstrate the feasibility and benefit of the PET approach with a prototype implementation developed in software, executing on an instrumented Pentium M system. First, we present a detailed characterization of the system response to power management mechanisms to identify the timescales and magnitude of expected response to management decisions. Second, we illustrate PET operation with realistic workloads and usage scenarios, demonstrating that the prototype achieves the desired ranges of operation with dynamic run-time control.

1.3 Contributions

This research proposes and evaluates a novel approach to managing computing systems to enable performance under power, thermal, and energy constraints. We begin by studying dynamic and static microprocessor power consumption to understand opportunities for power management. We experiment with simulated models of microprocessor and cache power management techniques. Then, we perform a detailed characterization of power management mechanisms to identify the timescales and magnitude of response to management decisions in live hardware. We introduce a framework for managing multiple concerns: performance, power, energy, and temperature, in complex systems. We demonstrate the feasibility and benefit of the PET approach with a prototype implementation developed in software, executing on an instrumented Pentium M platform. We illustrate PET operation with

realistic workloads and usage scenarios, showing that the prototype achieves the desired ranges of operation with dynamic run-time control. Our research contributions include the following:

1. We developed a detailed power model of an Alpha 21264 processor in a simulator that tracks both energy used per instruction, and energy used for each physical component in the processor core. We analyzed the energy spent throughout the pipeline, and determined the extent of power wasted on misspeculation and over-provisioning structures.

2. We analyzed three leakage energy reduction techniques for on-chip caches. We found that state-preserving techniques were appropriate for the secondary cache due to the large penalty for re-acquiring data from off-chip memory locations, and that leakage-control mechanisms that increase cache access time could actually cause more energy expenditure due to longer program execution times.

3. We performed a detailed analysis of power and thermal behavior in response to DVFS with an Intel Pentium M system. We identified a two-phase thermal response to changing DVFS levels in the CPU temperature sensor, first with an immediate (tens of milliseconds) temperature change and then additional CPU temperature drift due to heating or cooling of the local ambient air.

4. We developed power models that predict power at all p-states based on observed data from the present p-state on an Intel Pentium M platform:

- performance counter-based power model for clock throttling and DVFS p-states
- performance counter-based power model for DVFS only
- measured power-based power model with SPEC CPU2000 `train` input set

5. We designed the PET framework, including the multi-dimensional goal specification, to enable a run-time manager to effectively control multiple concerns simultaneously.

6. We built prototype PET manager and evaluated the manager under realistic conditions on live hardware on an instrumented Pentium M system.

1.4 Organization

This document is organized as follows. Chapter 2 discusses prior work in the fields of power, energy, and temperature management, to provide context for my current research. Chapter 3 highlights our prior research that explains the nature of microprocessor power consumption, evaluates options for static power management, and identifies the effects of workload variability on power and performance management. In Chapter 4, we explain in detail the methodology for monitoring power, performance, and temperature of a Pentium M system. Chapter 5 follows with an analysis of performance, power, and thermal response to the available p-state mechanisms on the prototype platform: clock throttling and DVFS. Chapter 6 provides an overview of run-time, coordinated management for performance, power, energy,

and temperature, describing the PET framework architecture and key components for effective management. Chapter 7 demonstrates a prototype PET manager implementation. We show the feasibility of expressing goal targets and tuning the system response at run-time, with reasonable overhead and accuracy. Chapter 8 concludes with a summary of research contributions and a discussion of the PET approach.

Chapter 2

Related Work in Power and Thermal Management

Semiconductor integration presently allows more transistors and wires fabricated per processor die than contemporary power distribution and cooling technology could handle without intervention from power and thermal management techniques. Several solutions for power and thermal management have been deployed in practice or explored through research simulation. In this chapter, we introduce underlying mechanisms for manipulating power consumption, execution time, and heat output. Then, we discuss a collection of techniques that employ the mechanisms throughout the computing software-hardware stack.

2.1 Power and Thermal Management Mechanisms

2.1.1 Throttling

Pipeline throttling was one of the first thermal-management mechanisms on modern microprocessors. Sanchez *et al.* described the Thermal Assist Unit (TAU) on a PowerPC processor in which thermal sensors continuously record temperature values during operation [79]. When the measured temperature exceeds a thermal threshold, the TAU launches instruction-cache throttling, a form of pipeline throttling that reduces activity throughout the pipeline by restricting the instruction stream.

A more extensive form of throttling, clock throttling, controls activity throughout the whole chip by periodically interrupting the CPU clock, reducing average power and allowing the chip to cool between bursts of activity. Throttling is drastic in the Pentium 4 `STOP_CLK` mechanism that allows only a small burst of CPU activity between long cooling periods; it severely degrades performance to ensure safe thermal conditions [45]. Other processors allow a range of software-selected throttling levels for a finer granularity of performance, power, and temperature control. The Pentium M processor employs two grades of clock throttling. The software-controlled version sets user-defined throttling levels that can be adjusted dynamically to tune power and temperature levels, at the expense of performance degradation proportional to extent of throttling. A hardware-only control mechanism relies only on thermal sensor input, overriding user selections to protect circuitry from overheating in the event of temperatures exceeding pre-set thresholds [77].

2.1.2 Leakage Control

As fabrication technology generations approached the 100nm node, transistor subthreshold and gate leakage became a serious issue for power consumption. Transistors allow current to flow through the device even while in a nominally "off" mode. The on-chip memory hierarchy of caches consume a large percentage of chip area, and with greater transistor density than random logic. Thus, controlling leakage power in caches became a high priority. Researchers investigated several mechanisms in an effort to curb the growing power problem. One method is to fabricate transistors with combinations of high threshold voltage (V_T) transistors

with low threshold voltage devices. The high-V_T transistors are slower and leak less current, while the low-V_T transistors are faster and leakier. Tschanz *et al.* reduced leakage current by 20% for a 130-nm technology with a mix of high and low V_T devices [84]. The Gated-V_{DD} approach adds an extra transistor between leaky circuits and either the power supply (V_{DD}) or electrical ground, to restrict the current flow. The additional transistor is fabricated with a higher threshold voltage for less leakage than the circuits it protects [70]. The Gated-V_{DD} mechanism curbs power; however, it does not preserve the memory contents, and thus cache lines must be re-fetched if they are prematurely discarded. An alternative approach is to modify the back-gate bias to dynamically increase the threshold voltage, reducing leakage, or reduce the threshold voltage to increase circuit switching speed [56], [68].

2.1.3 Reconfigurable Microarchitecture

Reconfigurable microarchitecture further extends the resizing concept to other structures on the chip, strategically reducing power consumption in components that are responsible for substantial fractions of the processor power, such as the issue queue. Albonesi *et al.* survey several approaches to adaptive issue queue resizing. One predictive control scheme increases queue size when the controller determines that IPC is likely to improve with a larger queue. A reactive control approach resizes the queue based on two queue statistics, the number of valid entries and the number of idle dispatch cycles due to a full issue queue. A hybrid approach down-sizes the queue reactively, based on issued-instruction age, and automatically up-sizes the queue periodically, predicting that increasing the queue at

regular intervals will help prevent performance degradation [4].

2.1.4 DVFS

Dynamic voltage and frequency scaling (DVFS) provides an effective way of tailoring power resources to current demands. Rather than alter the available capacity of microarchitectural structures, DVFS alters clock frequency and supply voltage supplied to the circuits. The Advanced Configuration and Power Interface (ACPI) defines protocols for power management states, *p-states*, in which the processor is operating at a reduced power and performance state [39]. Clock throttling and DVFS settings are typically used as p-states, with DVFS becoming more popular due to its superior power-performance efficiency. DVFS is widely available in commercial systems, including implementations in AMD's PowerNow [2] and Intel's Enhanced SpeedStep [46] and dual-core Dynamic Power Coordination [65]. Presently, most commercial systems apply a single DVFS level to all cores in one chip or multi-chip module. Researchers are investigating means to exploit heterogeneous voltage and frequency levels to adapt the operating regime to workload and performance needs.

2.2 Power and Thermal Management Approaches

Opportunities abound for adapting performance, power, energy, and temperature throughout software and hardware layers in computing systems. Table 2.1 lists both benefits and disadvantages of controlling power and temperature at each layer. This section explores ways in which the mechanisms previously described are applied throughout the software-hardware stack.

Layer	Advantages	Disadvantages
Application	preferred response embedded directly in workload	no run-time information available
Compiler	full view of program behavior	limited options for power or thermal management; no run-time information for static compilers
Out-of-band manager	does not require computation on main processors	large overhead for smaller systems
Operating system	view of both hardware and software	response time limited by scheduling quanta
Micro-architecture	target specific components	additional costly hardware verification and testing overhead; no programmability
Circuit	immediate response; tuned to individual chip's fabrication-process variation	no run-time re-programmability

Table 2.1: Summary of power and thermal management throughout the hardware-software stack.

2.2.1 Software

The compiler can analyze a broad view of program behavior and inject hints or commands into the program. Offline compiler-level power control can suggest voltage and frequency settings based on program information [40], [93]. The operating system can dynamically adjust DVFS levels in real time [1]. The workload program itself could also contain hints about power management, similar to existing approaches for software-managed caches. Just-in-time (JIT) compilers have been used to monitor sections of program code that cause excessive current spikes (di/dt, the change in current over time), and dynamically re-compile offending code sections to avoid dangerous behavior [35].

Operating systems can harvest a rich source of run-time information from both programs and hardware to tailor operation to present conditions. Kotla *et*

al., apply dynamic program classification and scheduling techniques to adjust the frequency and voltage of a system to minimize the power consumption while maintaining performance levels [55]. Flautner *et al.* developed Vertigo in a Linux operating system to coordinate several specific performance-setting policies that govern DVFS settings, using run-time information to predict the best policy to suit the current workload characteristics [20]. Brock *et al.* developed a Dynamic Power Management (DPM) architecture for embedded systems and prototyped DPM on a PowerPC 405LP processor in Linux [42]. DPM invokes policy managers to choose policies that map high-level operating states such as sleep mode or task states to pre-set low-level settings, including voltage and frequency, bus speed, etc., with the intent to reduce energy while meeting application performance requirements.

Operating systems also provide a means for coordinated control over multiple components within a system. The ECOSystem (Energy-Centric Operating System) project for mobile devices developed a model of *currentcy*, a term used to express both electrical current and currency. They abstract energy into a resource to spend on devices throughout the system, such as disk accesses or CPU cycles [96]. Felter *et al.*, explore *power shifting*, in which the operating system partitions a fixed power budget between processor and memory in server systems, dynamically adapting to workload behavior to provide power resources as needed according to memory access patterns and computation needs [18]. Weissel, *et al.* devised *energy containers*, a system of resource allocation within an operating system to track energy usage throughout a multi-core system, estimate temperature, and dynamically assign work to meet individual components' thermal budgets [90].

For multiple cores, Annavaram *et al.*, calculate energy per instruction (EPI) to distribute work among four low-frequency cores rather than a single higher-frequency core when sufficient thread-level parallelism is available to improve throughput and reduce power [6]. Ghiasi *et al.*, developed an operating systems scheduler for a multi-processor system with heterogeneous-frequency processors. The operating system schedules tasks on the processor which most closely matches the workload's ideal frequency setting, as determined by CPU and memory access characteristics [22].

Isci *et al.* evaluate several DVFS policies to maximize throughput (instructions per second) in chip multiprocessors (CMPs). They describe a power controller similar to our coordinated manager [30], [31], [54]; the DVFS policies are similar to single-core approaches to maximize performance within a power budget [62], [72], and extended to manage multiple cores in a CMP configuration. The authors simulate approximations of three DVFS p-states and present results for a 4-core CMP with two combinations of steady-power benchmarks from the SPEC CPU2000 suite: a group of four memory-bound benchmarks, and a group of 3 memory-bound benchmarks with one computation-bound benchmark. They determine that the best policy is one that calculates the expected sum of IPS across the four cores and configures DVFS p-states independently per core to maximize the combined IPS within a fixed chip-level power budget. They note that commercial CMPs are currently constrained to a single DVFS state for the full chip, which significantly degrades performance compared to individual DVFS states for each core.

In server systems, Ranganathan *et al.* employ a centralized, policy-driven

controller to dictate power levels on individual server blades primarily to meet total power budget constraints, with policies to minimize performance degradation, allowing heuristics tuned for performance expectations in service level agreements [76]. Cameron *et al.* developed *PowerPack*, a framework to measure and manage power in large-scale high-performance systems. By profiling applications and understanding their compute-bound and memory/communication-bound phases, PowerPack uses DVFS to lower CPU frequency at opportune times to save power and energy (and therefore, operating costs) with minimal performance impact [11].

2.2.2 Hardware

Microarchitectural-level techniques enable power and thermal management pinpointed to specific areas on the chip, which is especially useful to manage power density and the resulting hot spots on die, or target specific power-wasting components. Heo *et al.* explore the use of redundant microarchitectural components to migrate activity from one instance to another allowing the warmer areas of the chip to cool [37]. Chaparro *et al.* evaluated clustered micro-architectures for a form of activity migration called *cluster-hopping*, in which alternating redundant sections of the chip are placed into an idle, low leakage power mode with the Gated-V_T mechanism. Power dissipation and heat load is thus spread throughout the chip over time, allowing hotspots to cool [12].

Manne *et al.* use fetch gating to manage the amount of power spent on misspeculated instructions. They track the confidence level of branch predictions in a superscalar out-of-order pipeline and trigger pipeline throttling to stop fetching new

instructions when the number of in-flight instructions with low prediction confidence exceeds a threshold [60].

Srinivasan *et al.* evaluate dynamic thermal management algorithms with three microarchitectural adaptations: proportional fetch throttling, which interrupts instruction fetch periodically, instruction window resizing, and restricting issue width (accompanied by turning off functional units). They note that fetch toggling affects thermal status of the entire chip by decreasing pipeline activity, while resizing the instruction window and disabling functional units target local issues. Consequently, they found that microarchitectural adaptation was more effective than DVFS for managing hotspots in most cases [83].

Several techniques selectively place cache lines in low-leakage modes to save static power. The *drowsy cache* technique by Flautner *et al.* uses two voltage supply lines, one for awake mode at 1 V and a separate 0.3 V supply for drowsy mode [19]. Kaxiras *et al.* employ the Gated-V_{DD} mechanism for a *cache-line decay* technique to place cache lines in an idle mode after a counter indicates the cache line has been unused for a specified number of cycles [50]. Yang *et al.* introduce the Dynamically Resized I-Cache, *DRI-Cache*, which modifies the effective cache capacity by placing portions of the cache into an inaccessible low-leakage mode with the Gated-V_{DD} mechanism. The extent of the cache in the low-leakage mode at any time is determined by the cache miss rate; when miss rates exceed a threshold, a portion of the low-leakage cache transitions to active mode to increase capacity [95]. Zhou *et al.* also employ the Gated-V_{DD} technique for leakage control with *Adaptive Mode Control* that monitors miss rates and adaptively resizes the cache according

to the difference between cache misses caused disabling useful data in sleep mode and cache misses that would have naturally occurred due to data not present in the cache [97].

Circuit techniques can be used for low-level, automatic control for localized decisions with a fast response time. Clock gating, once used in a limited fashion [24], is now ubiquitous. Clock gating adds a logic gate between the global clock tree and local clock signals, allowing the clock signal routed to independent units to be enabled and disabled quickly, reducing switching power. Specialized circuits have been proposed to handle specific tasks. Flautner's *Razor* technique, for example, automatically adjusts supply voltage according to the error rate, pushing the envelope of correct execution to squeeze the best performance, while maintaining a safety net in case of incorrect execution [16]. Drake *et al.* report a critical path monitor (CPM) circuit on the IBM's POWER6 processor that monitors timing margin and process variation [15] that could be used to precisely tailor voltage levels to individual chips, accounting for manufacturing process variation.

2.2.3 Multiple Layers

With techniques for power and thermal management available throughout the hardware-software stack, researchers have investigated coordinating power, energy, or thermal management throughout multiple layers.

Adve *et al.* created the GRACE (Global Resource Adaptation through CoopEration) project that applies a coordinated approach throughout system layers to save energy in multimedia systems by adapting to workload behavior. They

use a hierarchy of infrequent global adaptation and frequent localized adaptations to minimize energy expenditure while meeting frame-rate deadlines in multimedia applications. The adaptations consist of layer-specific mechanisms, such as DVFS for the hardware (CPU) layer, reducing computation (quality) for the application, and adjusting the network bandwidth budget. By applying effective mechanisms at appropriate points in the hardware-software stack, and at appropriate timescales, they observed a synergy in energy reduction, in which the energy savings for local and global adaptations combined exceeded the sum of individual reductions from techniques applied in isolation [87].

Hazelwood *et al.* of the Tortola project aim to manage power and performance, as well as address concerns such as reliability and security, by a sharing information between layers. They focus on problems that can be detected by hardware and solved in a software virtualization layer, including their dynamic recompilation for avoiding di/dt problems [34], [35].

Kephart *et al.* coordinate separate power and performance controllers within an IBM Blade server. The performance controller functions within the Websphere Extended Deployment manager in middleware, and the power manager spans operating system, middleware, and software levels. A multi-criteria utility function expresses a relationship between power and performance, allowing optimization of both criteria according to user preferences. They experiment with a utility function to optimize "performance minus a weighted power cost," and identify alternative utility functions, such as optimizing "performance per Watt" or "performance, subject to a power constraint." They compare a machine-learning algorithm with a

handcrafted policy, and also with a free-running unmanaged case. The best-case machine learning policy saved more power than the handcrafted policy, while meeting web server response-time expectations [51].

2.3 Control

2.3.1 Predictive

Many of the aforementioned approaches rely on predictive control, comparing current indicators with expected behavior to choose the best policy, DVFS setting, task schedule, etc. Offline approaches must be predictive due to their static nature. Approaches that do have access to run-time information can still benefit from predictive control. Multimedia applications with fixed deadlines, for example, can predict an appropriate DVFS setting without the overhead of repeatedly changing settings with a high-overhead mechanism to reach a desired point [83].

2.3.2 Reactive

An alternative method is reactive control. Feedback loops based on control-theoretic techniques provide robust control, which is especially useful in situations where conditions are not fully known *a priori*. Skadron *et al.* apply well-known techniques from control theory, including the classic PID controller (proportional-integration-differential) to thermal management [81]. Juang *et al.* propose closed-loop control to coordinate DVFS levels for individual cores within a chip multiprocessor to match frequencies to the amount of work in parallel segments between software synchronization points. Cores with more work to do execute at higher speeds, while lightly loaded cores slow down to save energy [49].

A dedicated controller with closed-loop control can provide automatic management for complex tasks, without adding complexity to the operating system or executing applications. One solution designed for managing both power and temperature simultaneously is Intel's Foxton Technology, which employs a feedback control system to maximize processor frequency, and therefore performance, within a power-thermal envelope [69]. As the processor power consumption and temperature vary with workload and environment, on-die sensors provide information every 8 μsec to the embedded Foxton micro-controller, which first enforces the thermal limits, and if the processor is within a safe thermal zone, the controller raises or lowers frequency in small increments until it reaches a programmable power limit. If the minimum frequency and voltage settings cannot meet the limit, the microcontroller directs the primary processor into single-issue mode, and as a last resort, it alerts the operating system that it could not meet the limit. The team found that current draw was highly dependent on workload behavior; the closed-loop control provides an effective means of tracking present conditions and modifying DVFS settings to match workload demands while meeting power and thermal limits.

Lefurgy *et al.*, demonstrate closed-loop control on IBM BladeCenter servers using processor clock throttling to enforce power limits with a proportional (P) controller. Their technique achieves higher performance than two alternatives, an open loop approach of a fixed throttling setting that meets the power budget for all workloads, and a common simple closed-loop solution that raises or lowers the throttling level by one step depending on whether the measured power is greater or less than the desired set point to meet the power budget. Their P controller

technique was most advantageous for low power budgets, when extensive throttling is required to meet the power limit.

2.3.3 Effectiveness

The extent to which mechanisms are effective at power, energy, or thermal control depends upon their application. Li *et al.* investigated the differences between simultaneous multithreading (SMT) and chip multiprocessor (CMP) microarchitectural classes with respect to performance, energy, and temperature. They observed that the inherent differences in the microarchitecture lead to different types of thermal behavior: the sophisticated logic in SMTs created localized hotspots, while the simpler cores in CMPs are more evenly affected by overall energy expenditure. They also noted that due to the architecture-specific thermal patterns, localized thermal management techniques were more effective for SMTs and DVFS was more appropriate for CMPs [58]. Thus, it is important to match the needs of the system with the capabilities of a management mechanism.

2.4 Analysis

The body of related work illustrates a wide range of management techniques designed to cope with the growing problems of performance-aware power, energy, and temperature throughout several layers of hardware and software, from circuits and microarchitectural components to operating systems and compilers. Individual mechanisms, such as DVFS or pipeline throttling, are applied with strategic techniques, such as dynamic thermal management, to achieve a desired outcome.

Control types, predictive or reactive, determine the means by which the techniques interact with the system.

Research has demonstrated that different mechanisms, techniques, and control types fit different situations, and that judicious combinations of methods can yield synergistic results. Conversely, and not as widely reported, it would be possible for multiple individual techniques to attempt control of many individual mechanisms with destructive results, or even attempt to orchestrate a common mechanism with conflicting directives. For example, if the compiler, operating system, and hardware each chose DVFS settings for the same component, there would be no guarantee that each layer would agree on the same setting.

An intelligent management system could selectively enable a collection of techniques and mechanisms, benefitting from positive interaction with a coordinated approach. For example, the operating system could use a compile-time DVFS technique to read compiler annotations for initial voltage and frequency settings, then to adjust the settings based on run-time information from temperature sensors. In our work, we seek to exploit information available throughout the hardware-software stack, and then make coherent decisions *at the appropriate level* for response time and control communication bandwidth. Rather than micro-managing DVFS levels from a high level, we allow those layers with a global view of the application and system as a whole to synthesize information into goals for lower levels. The lower levels then translate the goals into specific objectives for performance, power, energy, and temperature, and make localized decisions on a faster time scale that can be tailored more closely to the executing workloads.

Chapter 3

Power Modeling and Management

In previously published works, I have contributed to microprocessor power modeling and management studies [28], [29], [33], [48], [67], [72], [73], [74], [75], [85], [86], designed logic for the TRIPS prototype operand network [25], [80], and introduced the PET approach to coordinated management to the research community [30], [31], [32].

In this chapter, we first track sources of power consumption and identify areas of wasted power [67]. Second, we investigate static power management techniques, comparing three approaches to saving leakage power in SRAMs, which comprise a large portion of the microprocessor chip area [28], [29]. These two studies indicate that a substantial portion of microprocessor power supports features that are not necessary for correct computation, but rather provide performance-enhancing capabilities, such as large cache capacities and wide-issue core pipelines. The power models and management techniques in these studies suggest that microprocessor power could be managed to provide resources on demand, a concept that we developed further with the PET manager.

Third, the inherent variation in computing systems poses challenges to effective management. We quantify the variation in both hardware and software to better understand the nature of the system response. The extent of variation sug-

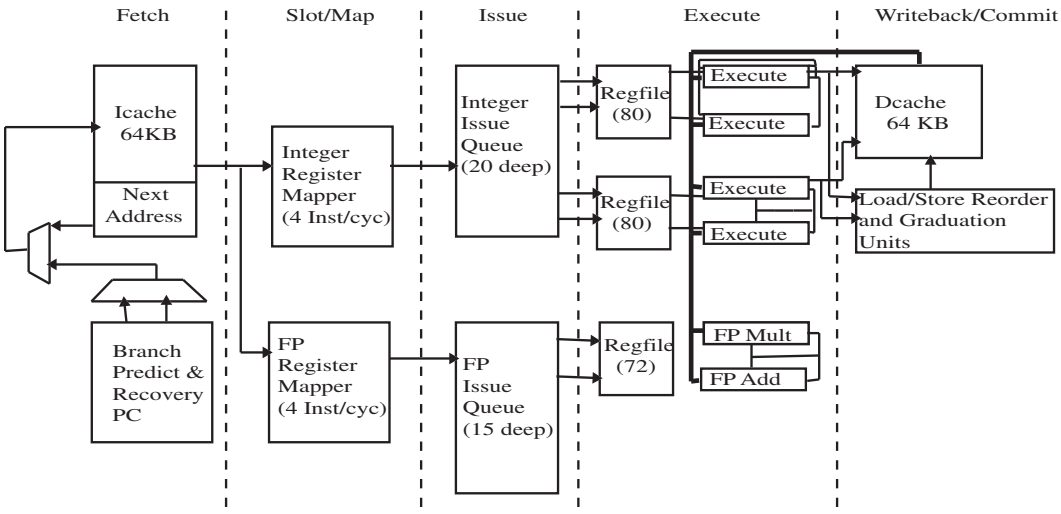


Figure 3.1: Alpha 21264 pipeline diagram [adapted from Gieseke *et al.* [23]].

gests that a dynamic run-time management scheme tailored to present conditions will be effective, whereas *a priori* policies may not guarantee the desired response.

3.1 Dynamic Power

We examined the dynamic power consumption profile of a high-performance superscalar microprocessor, focusing on two categories of microarchitectural features used in high-performance microprocessors that contribute to bottom-line performance at the cost of substantial power use: speculation and over-provisioning [67].

Superscalar microprocessors rely on speculation to feed their wide issue, out-of-order, and deep pipelines. Control speculation, data dependence speculation, hardware pre-fetching, cache way prediction, pipeline scheduling speculation, and other predictive mechanisms allow the processor core to make forward progress with-

out waiting for long-latency operations to complete. Speculation offers opportunities for saving energy by filling the pipeline with useful work to do, thereby increasing throughput and reducing the program execution time. With fewer idle cycles spent resolving cache line addresses or branch targets, for example, the processor could finish tasks earlier, using less static power and allowing more opportunity to transition to a lower-power mode.

However, speculative techniques also cause a power burden from effort wasted on mis-speculated instructions. In addition to predictor structure control logic and arrays, speculative features also require additional resources throughout the chip, effectively providing extra room in the pipeline for extraneous instructions. Each instruction that is ultimately discarded contributes indirectly to elevated power levels due to the need for increased structure sizes, which lead to higher levels of transistor leakage current and more signal capacitance. A useless instruction also directly affects dynamic power through datapath switching activity until it is ejected from the pipeline.

The second microarchitectural feature in the study is *over-provisioning* that results from excess capacity in the pipeline. Over-provisioned hardware structures have a wide range of effects on microprocessor power and energy consumption. Under-used array capacities and read/write ports cause the array to consume more dynamic power than necessary for signal switching in larger decoders and longer wordlines and bitlines. Furthermore, the excessive number of transistors contribute to greater leakage current. Secondary effects of over-provisioning include longer interconnects to route the overly large units and possibly increased temperature

levels from additional switching and leakage currents.

3.1.1 Microarchitectural Simulator

We modeled an Alpha 21264 processor by combining the architectural simulator `sim-alpha` [13] with power components from the Wattch [10] power estimator, and adding monitoring probes within the simulator to observe structure accesses and the full path of each instruction through the pipeline.

`Sim-alpha` models many low-level hardware features that support speculation throughout the pipeline. The cache model includes cache line and associative way prediction, and optimistically issues loads and stores as if there were no address conflicts and no port contention. The fetch unit uses a tournament branch predictor to speculatively determine the direction and target address of branch instructions [52]. The simulator also models trap detection and recovery, including clearing the pipeline and re-fetching instructions.

We augmented Wattch's power model with additional components, such as I/O pins and an estimate of bus and system interface power, and adapted the model for Alpha-specific features with a datapath width of 64 bits and processor frequency of 600 MHz.

We designed the power model to produce power levels similar to published data [24], [61], [91]. Our baseline power rating with all units consuming full power is 71 Watts. However, in our study, we separate the pipeline into structures that consume constant power each cycle regardless of the number of incoming instructions, such as map logic and issue queues, and other components such as ALUs

Component	Energy Per Cycle (nJ)	Average Power (W)
Clock	38.33	23.00
System	6.53	3.92
	Energy Per Access (nJ)	Average Power (W)
Fetch	3.49	6.74
FP mapper	1.05	2.51
FP issue queue	0.60	1.50
FP ALU	3.58	0.0
FP register file	1.67	0.0
Integer mapper	1.55	3.72
Integer issue queue	2.06	4.95
Integer ALU	2.33	2.34
Int register file	2.51	8.48
Load and store queues	4.25	0.98
Data cache	10.00	2.210
gzip Total:		64.48

Table 3.1: `sim-alpha` power model, with average power for `gzip` benchmark.

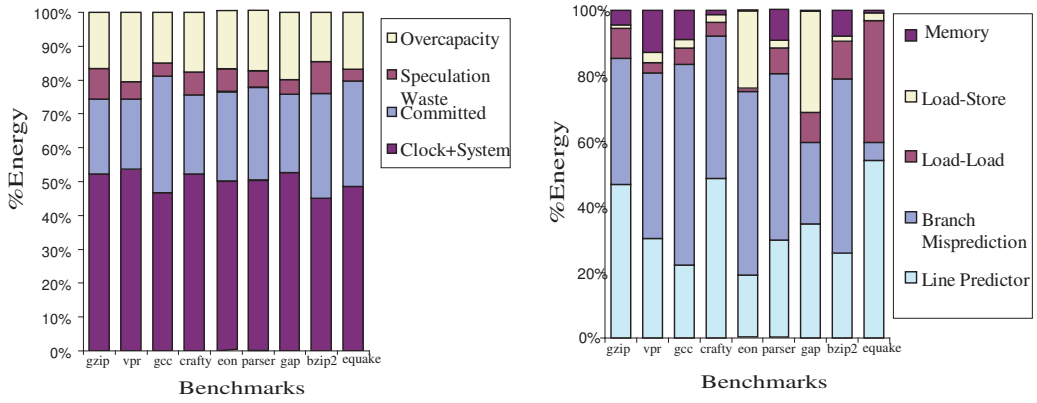


Figure 3.2: Energy expenditure: (a) overview (b) speculation energy by category.

and register files that vary in power consumption depending upon instruction activity. For the power-variable structures, we make the simplifying approximation that structures consume a fixed amount of power per instruction and are effectively clock-gated when idle; for example, an adder consumes the same amount of power for each add operation, regardless of the operand bit values. For the benchmarks in this study, our model produces power levels ranging from 54 to 62 Watts, reflecting reduced power due to clock-gating and limited use of the floating-point cluster.

Table 3.1 lists components of the power and energy model. The second column shows the breakdown of energy per cycle for global components and energy per instruction for individual structures. To calculate energy use, we multiply the count of structure accesses by the power cost per access for individual components. We multiply the clock and system (including bus interface units and package pins) power costs per cycle with the program length for the global-structure energy total. The total energy is the sum of individual and global structures; average power is the total energy divided by program length. The third column in Table 1 shows results for average power for a representative program, `gzip`.

Our benchmark suite consists of several programs from the SPEC 2000 suite that represent a range of application behavior: `gzip, vpr, gcc, crafty, parser, eon, gap, bzip2` and `equake`. We simulated each benchmark for a total of one billion committed instructions after fast-forwarding through initialization code.

3.1.2 Pipeline Monitoring

To gain insight into pipeline over-provisioning and speculation, we monitor the simulated pipeline by two simultaneous methods. We track each instruction's path through the pipeline and keep a record of its hardware structure accesses. Meanwhile, we compile histograms of accesses to each major structure in the pipeline. With these two measurements, we are able to observe pipeline utilization for programs in the benchmark suite, evaluate speculative mechanisms' ability to fill the 4-wide pipeline with useful work, and determine the power overhead of speculation and over-provisioning.

Speculation: We monitor the power overhead of speculation by observing each instruction throughout the pipeline. When an instruction exits the pipeline, its hardware access record is classified into one of six categories according to the reason for termination. The categories separate work performed by the processor core into useful work for committed instructions (COM) and distinct causes of wasted effort: branch mispredictions (BR), cache line predictor mispredictions (LP), load-load conflicts (LL), load-store mis-speculation (LS), and memory traps (MEM). For example, an ADD instruction ejected from the pipeline when another instruction triggers a memory trap would contribute its access history to the MEM category even though it did not cause the trap because effort expended on its behalf was wasted by the memory mis-speculation.

At the conclusion of program simulation, power models applied to structure access counts determine the total energy, or power accumulated over time. We

maintain a separate energy account for supporting structures such as the global clock network, system interface, and I/O pins that are not attributed to individual instructions.

Over-provisioning: We monitor structure utilization by collecting the number of accesses to each structure, every cycle. Then, we apply our per-access power model and sum the structures' power use over the duration of the program execution to estimate the total energy consumed. Some microarchitectural structures, such as the integer and floating-point units and the caches can be designed to burn a negligible amount of power when they are unneeded. The calculated energy totals include contributions from these units according to the number of structure accesses, with no penalty calculated for over-provisioning. However, other structures are typically accessed every cycle, regardless of how many instructions actually use them. This class of structures is designed with sufficient capacity and ports to handle peak throughput, but under typical loads add an excess power burden to the pipeline. For example, the floating-point mapper and issue queue run continuously even during predominantly integer programs as they search the incoming instruction stream for useful work to perform.

In our model, we separate the power consumed by the integer and floating-point mappers and issue queues into power spent on instructions and power wasted by unused slots in the pipeline. The instruction power is categorized into useful and non-useful work; the empty-pipe power is accounted separately as a distinct power overhead.

Note that in the over-provisioning analysis, some portion of a chip's global clock network and supporting circuitry is over-provisioned due to the extra capacitive load and area of the over-provisioned structures; this study does not include the global structures in the utilization accounting.

We include only dynamic power in this study, based on our model of the Alpha 21264's 350nm process technology with negligible leakage current. In fabrication processes with larger leakage current, static power would significantly increase the penalty of unused and under-used structures throughout the pipeline.

Figure 3.2 charts the experimental results for overprovisioning and speculation. We found that power wasted by mis-speculation accounts for approximately 6% of the total energy, and power spent on under-used map and issue resources contributes about 17% of the total energy, considering only dynamic power and explicit power overheads in our model. We found that the front end of the core pipeline is most directly affected by speculation-related effort, subject to clearing and refilling to correct mis-speculation. The tail end of the pipeline has the advantage of containing fewer enqueued instructions subject to eviction upon a pipeline flush, and more information available from upstream stages for detection and power control of idle or under-used resources.

Our study results suggested that a power management policy that provides effective speculation and reduces overprovisionsing–such as adapting hardware resources to workload demands–could be highly effective in reducing the dynamic power and energy in a wide-issue superscalar processor. Dynamic power was the dominant source of power consumption in previous generations of process technol-

ogy due to low levels of leakage current and high capacitance. Current generations must consider static power, as well.

3.2 Static Power

High-performance transistors have become leaky in recent technology generations due to subthreshold and gate leakage current, and more transistors fit on each chip each generation, resulting in increased static power dissipation. A vast majority of transistors in modern microprocessors are used for on-chip storage.

In our static energy study, we explored the energy/performance trade-offs of three leakage-reduction techniques for on-chip level-1 and level-2 caches [28], [29]. This study used extrapolated values for circuit parameters available at the time in the year 2000, and considered only the subthreshold component of leakage current. In the elapsed time between our circuit simulations and the current date, new trends have emerged and circuit parameters have scaled differently than anticipated. We include this static energy study to demonstrate the sleep-mode techniques and our analysis, noting that although our projected current values deviate from contemporary process values, cache sleep modes are relevant for contemporary and future generations of processors.

We evaluated static power management techniques for on-chip caches, noting the control techniques' effectiveness in energy reduction and effect on processor performance [29]. We compared three techniques against a high-performance cache without leakage control.

One method, *dual-*V_T, employs slower transistors with a higher threshold

voltage, and hence lower leakage, in SRAM arrays. Transistors in the remainder of the cache circuit have a lower threshold voltage for faster switching speed. This dual-V_T method decreases subthreshold leakage currents but increases the cell access time compared with an SRAM composed of fast, leaky transistors [63],[78].

Another method dynamically adjusts the effective size of the array by employing the *Gated-V_{DD}* mechanism. In this scheme, a low-leakage transistor is used to selectively shut off the power supply to a subset of SRAM cells [70]. Thus, the capacity of the array adjusts dynamically as the amount of active information in the cache changes throughout the duration of the program. In our implementation, we use one NMOS gating transistor between a cache line and the ground node, which causes the memory cell to lose stored contents during the low-leakage mode. Our approach mirrors the *cache decay* technique, which also uses Gated-V_{DD} to turn off cache lines which have not been used in a specified amount of time.

A third technique, *MTCMOS*, dynamically changes the threshold voltage by modulating the backgate bias voltage [59],[68]. With this technique, memory cells can be placed into a low-leakage "sleep" mode yet still retain their state. Cells in the active mode are accessed at full speed while accesses to cells in the sleep mode must wait until the cell has been awakened by adjusting the bias voltage. While the MTCMOS technique has been implemented for an entire SRAM [68], we examine this idea using fine-grain control of each cache line.

In our experiments, we explored the energy and performance trade-offs of these techniques. Figure 3.3 summarizes the energy and energy-delay product (EDP) for each configuration. Each technique is effective in reducing energy con-

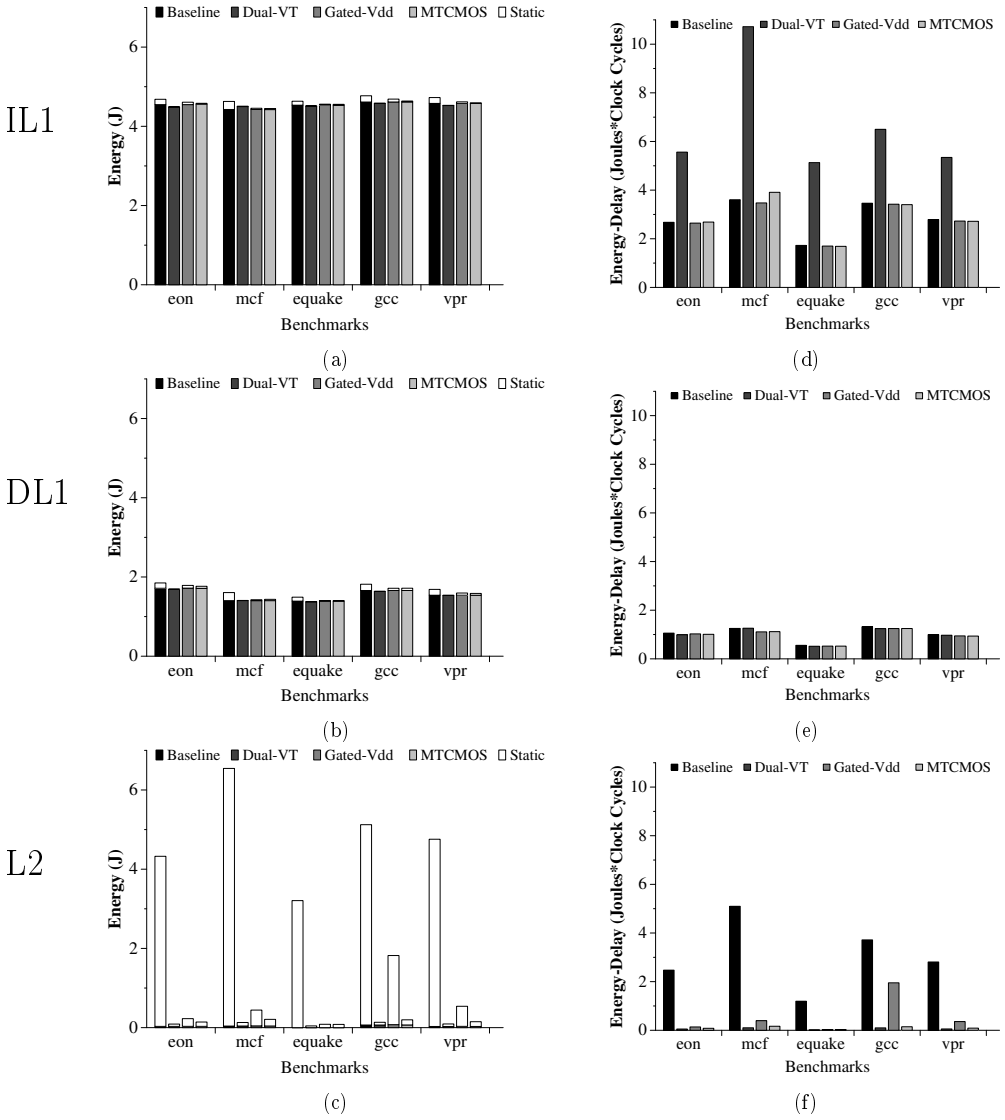


Figure 3.3: Energy and energy-delay product for L1 and L2 caches.

sumption in primary and secondary caches. We found that with careful selection of decay intervals, the MTCMOS and Gated-V_{DD} techniques yielded better energy-delay products than the dual-V_T technique in the primary caches, due to their overall lower access time.

With our assumptions, both the Gated-V_{DD} and MTCMOS techniques improve the energy-delay product by 2% in the IL1 cache, and yield an improvement of 6% and 7%, respectively, in the DL1 cache compared to the experimental baseline. The dual-V_T technique improves the energy-delay product of the DL1 by 4%, and degrades energy-delay product in the IL1. For the secondary cache, the dual-V_T technique has the best energy-delay characteristics, with a 50-fold improvement compared to the baseline case. The Gated-V_{DD} and MTCMOS techniques were also effective at improving the energy-delay of L2 caches, with overall reductions of factors of 20 and 34, respectively.

We also investigated the sensitivity of microprocessor performance and energy consumption to additional cache latency caused by leakage-reduction techniques. Additional latency and energy penalties contributed by the leakage reduction strategy can extend program execution time and increase static energy consumption, especially when applied to the primary instruction cache [26]. Increasing the dual-V_T IL1 cache access by two extra cycles results in performance degradation of 74%, and a 387% increase in static energy expenditure. For an MTCMOS IL1 with a zero-cycle decay interval, performance drops by 93% and static energy increases by a factor of 18 when the wakeup latency is ten cycles rather than one. In the level-1 data cache, the effect of additional access time was less detrimental. A dual-V_T DL1

with two additional cycles of access time reduces performance by 4% and increases static energy by 9%. An MTCMOS DL1 with a ten-cycle wakeup latency causes performance to drop by 31% with the shortest decay interval; longer decay intervals do not suffer such performance degradation. The unified level-2 cache is the least sensitive to additional delays, with a 2% dip in IPC for the dual-V_T L2 cache accompanied by a 2% increase in static energy; an MTCMOS L2 cache with the worst case of immediate sleep policy caused 8% reduction in IPC and 7% increase in static energy consumed.

These techniques target static energy reduction in cache memories while considering the effect on processor performance and total energy. Large memory arrays account for nearly half of the total transistor count and generally contain redundant state that can be retrieved from alternate locations if necessary, which allow aggressive energy-saving techniques. Instruction cache resizing [95], cache decay leakage control [50], drowsy caches [19], and the techniques we studied all manage static leakage by directing portions of the cache into a low-leakage mode.

3.3 Variability

Software and hardware variation in real systems create a challenge to effectively manage power consumption, motivating the need for effective run-time management.

3.3.1 Software Variation

Software variation has been recorded on many timescales, from microseconds to days. In this section, we illustrate software variation and the challenges it presents for dynamic power-thermal management. As demonstrated with the example of web server traffic over a twenty-four hour period for the 1998 Nagano Winter Olympics in Chapter 1, a power-management scheme must adapt to the changing workload to achieve throughput to meet workload demands without wasting energy.

We also examined software variability on a time scale of tens of milliseconds, a timescale useful for operating system scheduling decisions. We executed the full SPEC CPU2000 benchmark suite with the `reference` input set, at the maximum frequency of 2 GHz and measured power and IPC (instructions per cycle) at 10-15ms intervals. Figure 3.4 illustrates IPC for four SPEC CPU 2000 benchmarks. These benchmarks exhibit a range of behavior, from minimal variation in `art` to chaotic behavior in `gcc`. Between these extremes lie `ammp` and `gzip`. The IPC levels in `ammp` are variable, repeating at predictable intervals; `gzip` consists of a series of phases. Average IPC for these benchmarks would provide a representative indicator only for `art`; other benchmarks would require a time-varying indicator to ensure appropriate run-time decisions.

Software execution also varies for the same benchmarks from run to run. We executed the full SPEC CPU2000 suite twenty times, each at the maximum 2 GHz frequency. Statistics in Table 3.2 show that the variation between twenty iterations is small compared to the execution time, about 1.5 minutes variation for approximately 70 minutes of execution. The inter-iteration variation is not as dramatic as the

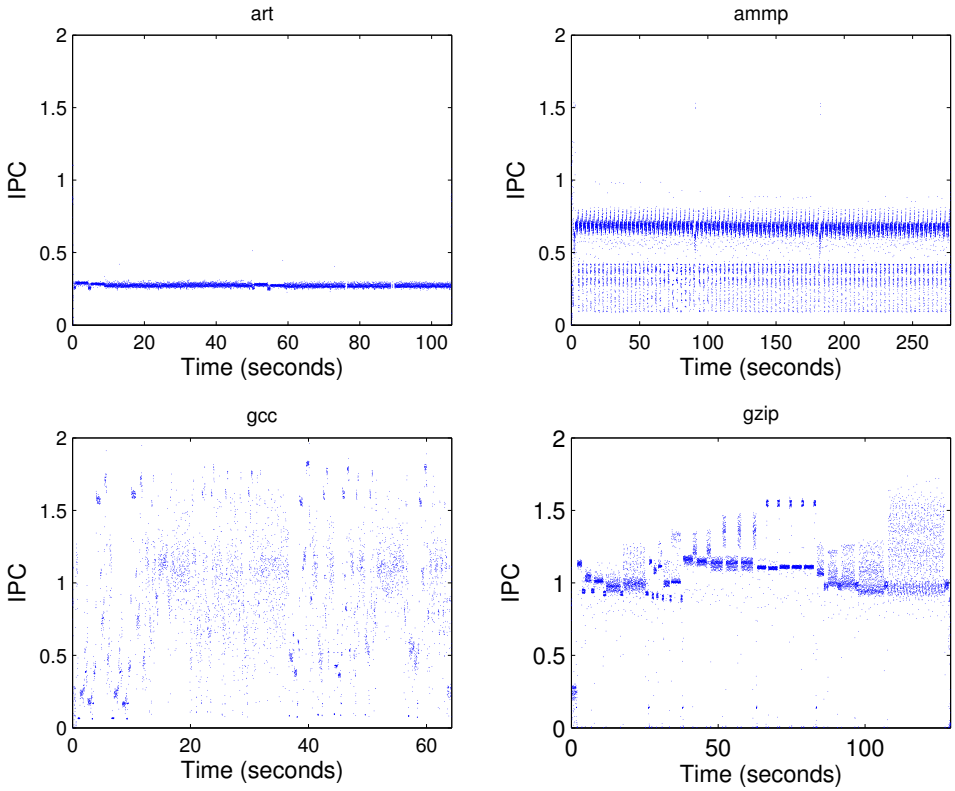


Figure 3.4: Performance (IPC) variation for 2 GHz operation in selected SPEC CPU2000 benchmarks: `art`, `ammp`, `gzip`, and `gcc`.

Execution Time Statistics	Time (seconds)
minimum	4220.7
maximum	4308.7
range	87.9
mean	4245.1
standard deviation	23.1

Table 3.2: Execution time statistics for twenty iterations of the SPEC CPU2000 suite at 2 GHz.

intra-variation for this suite, yet it is important to note that exact execution time is not guaranteed; power and thermal management approaches must be sufficiently flexible to accommodate fluctuations in execution time. Policies determined with offline profiling may not correctly project accurate execution durations.

Power consumption for the SPEC CPU2000 suite executing at 2 GHz on the Pentium M ranges between 3.5W and 16.9W, as shown in Figure 3.5. Although 95% of all samples fall between 10.9 to 15.9 Watts, the distribution of power levels through time is highly variable. Power consumption responds to workload phases and drops immediately to low power levels between benchmarks in the suite. Isci *et al.* found similar variation in power and execution time for the SPEC CPU2000 suite executing on a Pentium 4 platform [47]. Harnessing useful system information on a reasonable timescale will be essential to make intelligent management choices.

3.3.2 Hardware Variation

Another consideration is the variability of leakage current due to fabrication process variations: a 10% difference in gate length can cause a factor of three difference in a transistor's subthreshold leakage current [53], and process variation

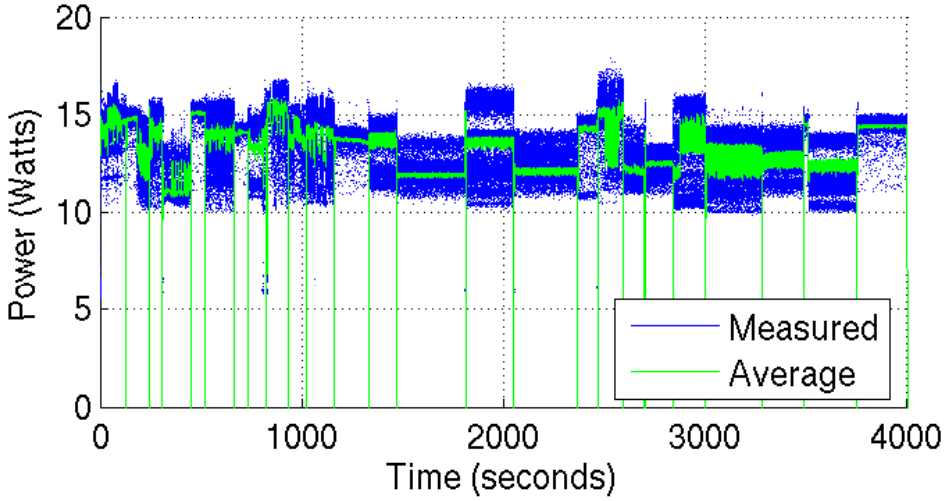


Figure 3.5: Power variation throughout SPEC CPU2000 suite execution.

is expected to increase in future process technologies. Process-induced variability in devices and interconnect influences chip-level power and thermal characteristics. Measured current in DRAM memory chips from five different vendors varied by a factor of two for active mode, and a factor of 1.5 for idle mode [33].

Within the same processor model, power can vary considerably. Rubio compared the average measured power for SPEC CPU2000 integer benchmarks executing on five Pentium M processors, nominally rated the same, under controlled conditions at 40 °C; the average power varies up to ten percent [33]. Thus, a power-management policy developed and tested for a specific architecture may not produce the same result on individual processors. An offline policy to enforce a strict power limit by pre-determined dynamic voltage and frequency (DVFS) settings would need to be conservative to accommodate the extra power for some individuals, at the

expense of needlessly curtailing performance for the lower-power individuals. In contrast, an adaptive policy that could detect the present power characteristics and tailor DVFS settings would allow the chips with power margin to run at higher frequencies, while still enforcing the power limit.

3.4 Summary

In our prior work, we have investigated the sources of power consumption, means of managing power, and the challenges inherent in dynamic power management. In this chapter, we present highlights from three studies.

Processor power consumption: Microarchitectural speculation, such as branch prediction, increases instruction throughput but carries a power burden due to wasted power for mis-speculated instructions. Pipeline over-provisioning supplies excess resources which often go unused. We applied our detailed performance and power model for an Alpha 21264 to measure both the useful energy and the wasted effort due to mis-speculation and over-provisioning. Our experiments show that flushed instructions account for approximately 6% of total energy, while over-provisioning imposes a tax of 17% on average. These results suggest opportunities for power savings and energy efficiency throughout microprocessor pipelines.

Leakage current management: Microprocessor performance has been improved by increasing the capacity of on-chip caches. However, the performance gain comes at the price of static energy consumption due to subthreshold leakage current in cache memory arrays. We compared three techniques for reducing static

energy consumption in on-chip level-1 and level-2 caches. One technique, dual-VT, employs low-leakage transistors in the memory cell. Another technique, Gated-V_{DD} can be used to turn off memory cells and discard their contents. A third alternative is dynamic threshold modulation, MTCMOS, which places memory cells in a standby state that preserves cell contents. Each leakage-control technique in the study effectively curbed leakage current with varying degrees of performance degradation.

Variation: Variation in both hardware and software poses a challenge to managing power on real systems due to unexpected responses to workload execution and power-management techniques. We investigated variation in several forms: intra-benchmark variation on a timescale of tens of milliseconds, suite execution time, workload response to power management with DVFS, and manufacturing variation that leads to individual characteristics for nominally identical components. Hardware and software variation through time and per component suggests that a dynamic run-time management tailored to present conditions will be more effective than an *a priori* policy that does not guarantee the desired response.

Chapter 4

Experimental Infrastructure

To explore the key features of our multi-dimensional performance, power, energy, and temperature (PET) management framework, we built a prototype software implementation that manages a single-core Intel Pentium M processor. The prototype platform provides the opportunity to observe software and hardware behavior with a processor architecture optimized for power management. The Pentium M architecture has been deployed in notebook computers, including the IBM ThinkPad series, and incorporated into high-density server systems such as IBM's Integrated xSeries servers [43], HP's ProLiant BL10e G2 [38], and Fujitsu-Siemens' PRIMERGY Blade BX300 [21]. This chapter describes the infrastructure of the instrumented Pentium M platform that serves as the test vehicle for the PET prototype, and illustrates the power monitoring features with a sensitivity study of measurement sampling granularity.

4.1 System Overview

In our experimental setup, the Pentium M system resides in a modified tower enclosure, lying flat with the panel removed to allow access for probe cables. The 90 nm Pentium M 755 (Dothan) processor on-chip memory consists of a 32 KB primary instruction cache, 32 KB primary data cache, and a 2 MB, 8-way unified

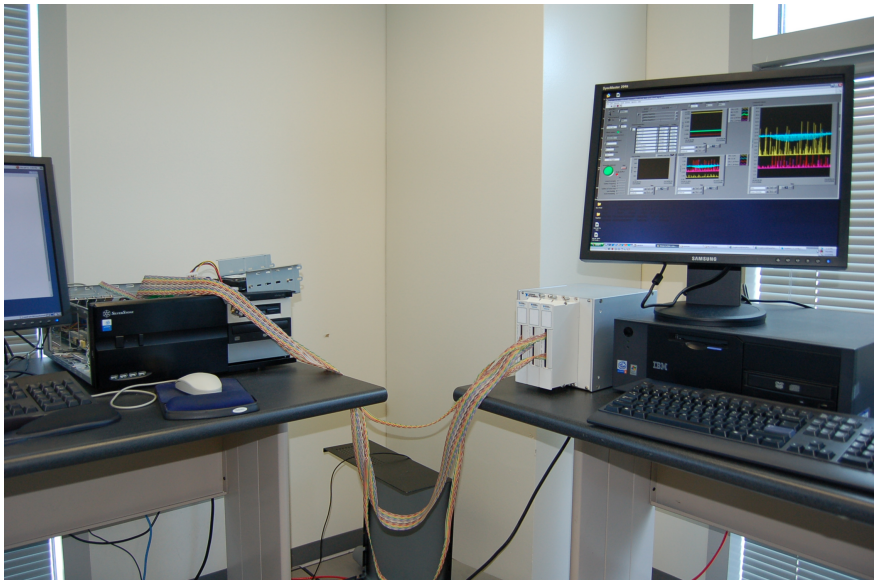


Figure 4.1: Pentium M (left), data acquisition module and measurement PC (right).

secondary cache [44]. The processor is paired with an Intel 855GME chipset and 512 MB of DDR SDRAM memory on a Radisys uniprocessor motherboard [71]. A Foxconn heat-sink and fan-assembly designed specifically for a desktop installation cools the Pentium M processor. With this arrangement, the processor operates throughout its full range of frequency, voltage, and throttling settings for all our experiments without triggering the processor's built-in throttling mechanisms for temperature overload [77]. One of two operating systems is selected upon reboot: Red Hat Linux or Windows XP. Unless otherwise noted, experiments described in this research use the Linux operating system.

Supply Voltage (Volts)	Frequency (MHz)
0.988	600
1.052	800
1.100	1000
1.148	1200
1.196	1400
1.244	1600
1.292	1800
1.340	2000

Table 4.1: Intel Pentium M 755 frequency and voltage pairs. Voltages selected from datasheet VID#A settings.

4.2 P-States

The Pentium M employs two power-management mechanisms: dynamic voltage and frequency scaling (DVFS) and clock throttling. We refer to DVFS and clock throttling settings as *p-states*, a reference to the ACPI variable performance states for power management.

The Pentium M supports supports 8 DVFS frequency-voltage pairs. We used the most conservative voltage settings, VID#A, for supply voltage in the range from 0.988 V to 1.34 V, with frequencies in 200 MHz steps from 600 MHz to 2.0 GHz, as listed in Table 4.1 [44]. We created customized drivers for the Pentium M to control DVFS settings via the Linux `cpufreq` function. Changing the DVFS setting incurs a stall up to 500μsec, a negligible overhead at 10-15 millisecond time scales for measurement.

The second power-management mechanism for the Pentium M is clock throttling. Clock throttling interrupts the main clock with a throttling signal to form

run and *hold* regions in the distributed clock signal. When the clock is enabled in the *run* portion, the clock signal runs freely with a standard clock period; when it is throttled in the *hold* portion, the clock is held at a zero-voltage level. During the *hold* period, the processor core and on-chip caches are idle, and consume a minimal amount of power. Eight clock throttling levels indicate the fraction of running time in increments of 1/8ths. For example, throttling level 8 is unthrottled (running 8/8ths of the time) and throttling level 3 runs 3/8ths of the time. The $run + hold$ window size is approximately 3 μs in the Pentium M.

While the effective frequency gained by clock throttling may be similar to DVFS, the influence on system behavior is fundamentally different due to the speed differential in the two mechanisms between the processor and off-chip memory. DVFS alters the ratio between the core clock and main memory frequencies. Clock throttling preserves the core-to-memory frequency ratio. Off-chip memory accesses issued during the *run* portion proceed while the core is idle in the *hold* time; however, with a large hold window such as in the Pentium M, in-flight memory accesses complete during the initial cycles of the *hold* time and the memory system is unused during the long period of remaining *hold* time.

4.3 Monitoring

We track performance and temperature directly within the Pentium M platform with performance counters and thermal sensors. In other commercial platforms, such as the IBM BladeCenter, power sensors are available on the motherboard; however, the Pentium M lacks integrated power sensors. We use an external

monitoring system to measure motherboard and CPU power, and calculate energy from power and elapsed execution time.

4.3.1 Performance Measurement

We track performance in two ways: the prototype software reports latency in terms of overall execution time and throughput as the rate of instruction completion according to the INST_RETIRED event counter that tallies the number of completed instructions. The Pentium M offers 96 possible events to monitor with the event counters, with only two events recorded at a time. In our experiments, we typically use one event counter to track INST_RETIRED for calculating instructions per cycle (IPC). With the other counter, we monitor DCU_MISS_OUTSTANDING, which is an approximate count of the number of cycles in which there are outstanding cache misses for the level-1 data cache. Although the count is imprecise, it does provide a view of idle time due to cache accesses, which affects both power and performance.

During benchmark execution, the software reads performance counter values with the RDPMC instruction at each sampling interval and saves the event values and timestamp in a buffer. The software generates an output file of timestamps and counter values at the conclusion of benchmark execution. Sampling interval length varies slightly, with most samples within 10-15ms and a mean sample length of 13ms.

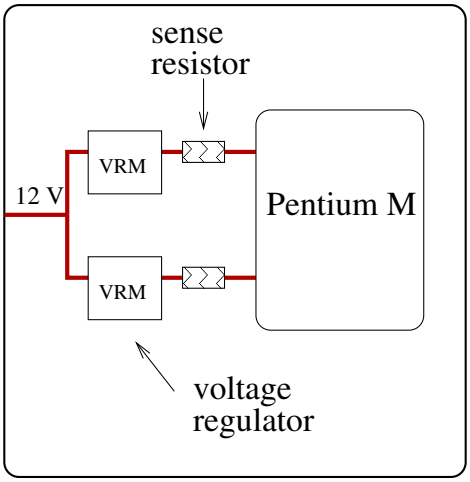


Figure 4.2: Diagram of voltage regulators and sense resistors.

4.3.2 Power Measurement

For highly accurate power measurement, we added two high-precision resistors to the power distribution circuitry, one resistor between each of two voltage regulator modules and the processor as illustrated in Figure 4.2. We tapped the high-precision resistors between each of voltage regulator modules and the processor with a voltage probe, providing voltages to a National Instruments data acquisition system that monitors processor supply voltage, and also calculates supply current via the voltage drop across the sense resistors. Figure 4.1 shows the Pentium M system on the left connected to the power-data acquisition system on the right. The data acquisition system interfaces with a Pentium 4 that executes a custom program in LabView software to capture the data in a trace file or send UDP packets of measured data to the Pentium M. The maximum voltage and current sampling rate is 333KHz; we typically use 80 - 100 voltage and current samples per second in

our experiments to match the performance sampling rate.

The monitoring software executing on the Pentium M raises a general-purpose I/O signal (GPIO) at the beginning of a benchmark and lowers the signal at the conclusion of the benchmark. The LabView software reports the value of the GPIO marker along with the power sample to facilitate alignment of the power trace from the Pentium 4 measurement machine with the Pentium M performance trace.

In power-feedback mode, the GPIO maker can be used to determine the time delay for the feedback loop. During benchmark execution, our monitor/control software queries the network socket for power data packets, one packet per sampling interval. The nominal sampling interval is 100 samples per second, however the actual sampling interval length on the Pentium M varies slightly, with most samples within 10-15 ms and a mean sample length of 13 ms, approximately 80 samples per second on average. The sample interval variation is periodic, with many consecutive samples near 10ms in duration followed by a long sample interval. In contrast, the sampling interval in the power-measurement software in LabView is steady at 10ms per sample. The mismatch in sampling rates creates excess power samples, which are periodically dropped within the UDP buffer as the queue of packets is overwritten with fresh data packets before all existing packets have been read. In situations where continuous measurement without dropped samples is desirable, we reduce the UDP packet transmission rate to 80 samples per second. In situations where lossy transmission is tolerable, we use 100 samples per second for better resolution.

For each time interval, the data acquisition reports current and voltage for

Probe Points	
1	motherboard power supply
2	GPIO pin used as benchmark duration marker
3	regulated 12-V supply
4	VRM1 output to processor
5	VRM2 output to processor

Table 4.2: Probe points for voltage probes. LabView software calculates current from voltage drop across resistors. Power-management software calculates processor power from VRM1 and VRM2 current and voltage measurements.

the voltage-regulator module (VRM) pins listed in Table 4.2. Both VRM1 and VRM2 outputs serve the Pentium M processor; the VRM's are paired with both set to the same p-state; however the voltage and current for each varies slightly due to differences in current draw and voltage drop.

4.3.3 Thermal Measurement

We collect two temperature measurements: the CPU temperature, for which a sensor is located within the processor chip package, and the ambient temperature, which is the motherboard temperature near the fresh-air intake vent. A thermal diode in the processor package is connected to an external A/D (analog-to-digital) converter that translates the measurements into junction temperature at a resolution of 1 °C. The prototype software queries both CPU and ambient temperatures from the LM85 fan controller chip via the system management bus, SMBus. We found that a blocking read for temperature queries delayed the monitoring software. We decoupled the sampling rates for temperature, and allow one temperature sample per N power and performance samples. Typically, we use a rate of one temperature query per ten performance-power samples. For detailed temperature analysis, we

use one temperature sample per two performance samples.

4.4 Benchmarks

4.4.1 Memory-Sensitive Microbenchmarks

We characterized application behavior for performance and power with a suite of microbenchmarks, *MS-Loops* [73]. The suite contains four microbenchmarks, each consisting of a small kernel of code that repeats multiple times for a given data working set size. We use 3 working sets, one for each level of memory hierarchy in this system. A 4-KB data footprint fits within the level-1 data cache, a 128-KB footprint resides in the level-2 cache, and a 4 MB footprint is too large for the on-chip caches, and must be serviced by accesses to the main memory. The 4 KB footprint is *compute bound*, meaning that computation speed determines performance. The 4 MB footprint is *memory bound*, with performance limited by waiting for accesses to main memory. The 128-KB footprint is intermediate in size; since it is serviced by fast L2 cache accesses, it is more similar to compute-bound than memory bound. Table 4.3 describes each microbenchmark in the `MS-Loop` suite. The application behavior for each benchmark is monophasic, which allows the benchmark to be accurately described by the average value of recorded data for power and performance.

4.4.2 SPEC CPU2000 Suite

We executed the full SPEC CPU2000 (floating-point and integer) suite with a fixed p-state for the duration of the run, for each of the 8 p-states. The official SPEC rating method would first perform an initialization routine for all benchmarks,

Microbenchmark	Language	Description
DAXPY	C	double-precision calculation of aX + Y
FMA	C	floating point multiply and accumulate
MCOPY	C	copy arrays from one memory location to another
MLOAD_RAND	C	random memory accesses

Table 4.3: Microbenchmarks: MS-Loops [73].

then launch each benchmark in succession. For logistical reasons, we split the suite into individual benchmarks and captured each trace file separately, leaving a short period between each benchmark invocation in which data are not recorded. Table 4.4 lists each benchmark in the SPEC CPU2000 suite in execution order, along with the mean IPC at 2GHz, the programming language, and benchmark description. Note that the mean IPC does not capture the time-varying characteristics of performance within a benchmark. The mean does does provide an indirect, aggregate indication of the level of pipeline activity during benchmark execution.

4.5 Sensitivity to Sampling Rate

We employ the instrumented system and benchmark suite to investigate key parameters for power and thermal control. In this section, we investigate the effects of the measurement sampling rate by varying the duration of sampling intervals. First, we executed SPEC CPU2000 benchmarks and recorded power data at a rate of approximately 80 samples per second, about 12 ms per sample. We used MATLAB software to aggregate samples and analyze the resulting trace data. We wrote a

Benchmark	INT or FP	Mean IPC at 2 GHz	Prog. Lang.	Description
gzip	INT	0.63	C	file compression
vpr	INT	0.40	C	FPGA placement and routing
gcc	INT	0.65	C	compiler for C language
mcf	INT	0.27	C	combinatorial optimization
crafty	INT	1.32	C	chess game playing
parser	INT	1.64	C	word processing
eon	INT	1.11	C++	visualization
gap	INT	0.44	C	interpreter, group theory
perlbmk	INT	0.95	C	perl programming language
vortex	INT	0.50	C	object-oriented database
bzip2	INT	0.99	C	file compression
twolf	INT	1.20	C	place and route simulator
wupwise	FP	1.12	F77	physics: quantum chromodynamics
swim	FP	1.44	F77	shallow water modeling
mgrid	FP	0.53	F77	multi-grid solver
applu	FP	0.32	F77	partial differential equations
mesa	FP	1.31	C	3-D graphics library
galgel	FP	0.88	F90	computational fluid dynamics
art	FP	0.27	C	image recognition, neural networks
equake	FP	1.70	C	seismic wave propagation simulation
facerec	FP	1.02	F90	image processing: face recognition
ammp	FP	0.18	C	computational chemistry
lucas	FP	1.23	F90	number theory, primality testing
fma3d	FP	1.25	F90	finite-element crash simulation
sixtrack	FP	1.10	F77	high-energy nuclear physics accelerator
apsi	FP	0.94	F77	meteorology, pollutant distribution

Table 4.4: SPEC CPU2000 benchmark attributes. Mean IPC measured at 2 GHz on an Intel Pentium M. Programming languages: C, C++, Fortran-77 (F77), and Fortran-90 (F90). Workload programming language and descriptions provided by Standard Performance Evaluation Corporation [82].

MATLAB program to read the original trace file, sampled at a mean of 12 ms per data point, and group multiple original samples into larger intervals to form a trace with a longer effective sampling interval. In the grouping, we average power across multiple original samples and accumulate the number of cycles. Grouped sets of 8, 40, 80, 400, and 800 samples create traces with approximately 100 ms, 500 ms, 1 second, 5 second, and 10 second measurement intervals, respectively. Then, we analyze the differences in observed behavior throughout a range of sampling interval sizes.

4.5.1 Power Sensitivity to Sample Interval Length

Figure 4.3 illustrates the sensitivity of power on sampling interval length for SPEC CPU2000 benchmark suite. In this experiment, the DVFS setting is fixed for all benchmarks at the 1800 MHz p-state. Strip charts from top to bottom in the figure show successively larger observation intervals, beginning with the top chart of the measured data at a mean interval time of 12 ms per sample, then 0.1-second, 1-second, and 10-second intervals. Charts for individual benchmarks `ammp`, `art`, `lucas`, and `wupwise` follow in Figures 4.4 through 4.7. As the charts indicate, longer measurement observation intervals exhibit attenuated power fluctuations as high and low values average together. We examine the effects of observation interval time in closer detail in the following sections.

4.5.2 Workload Sensitivity to Sample Duration

Figure 4.8 shows the effect of observation interval length for four SPEC CPU2000 benchmarks. Each chart plots the distribution of power values with curves

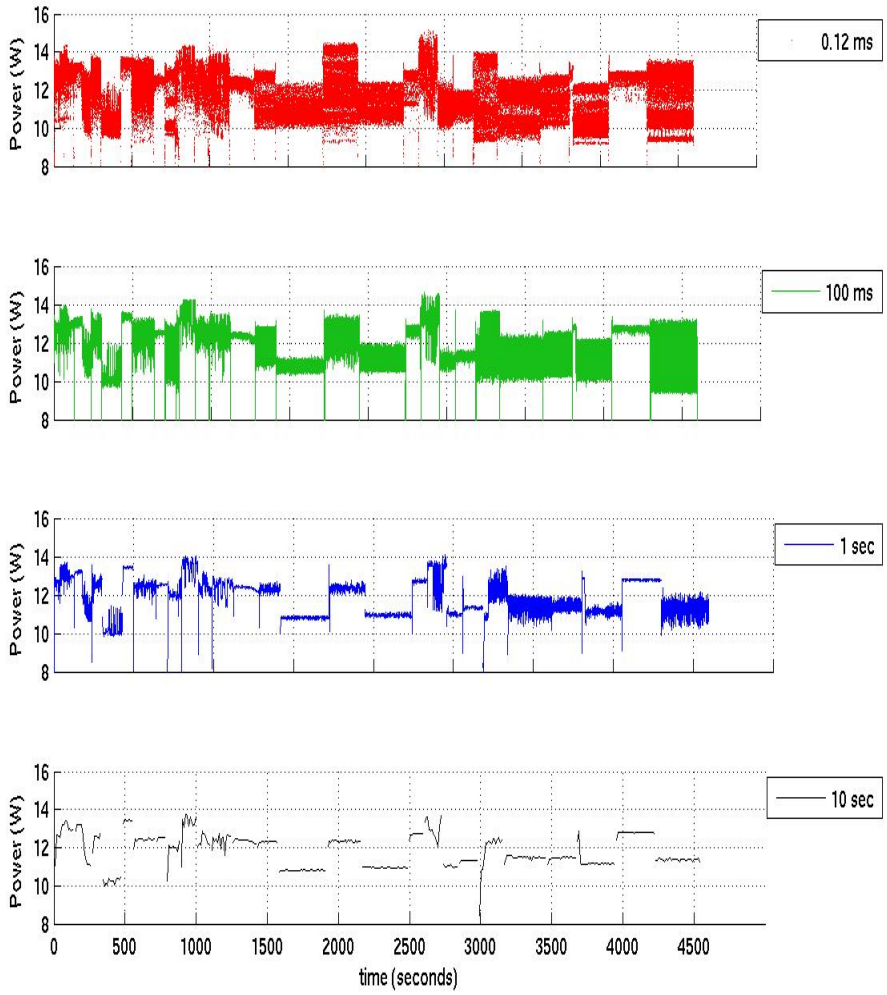


Figure 4.3: Average power per sample for the SPEC CPU2000 benchmark suite at multiple sample-interval lengths, from 12 ms per sample to 10 seconds per sample.

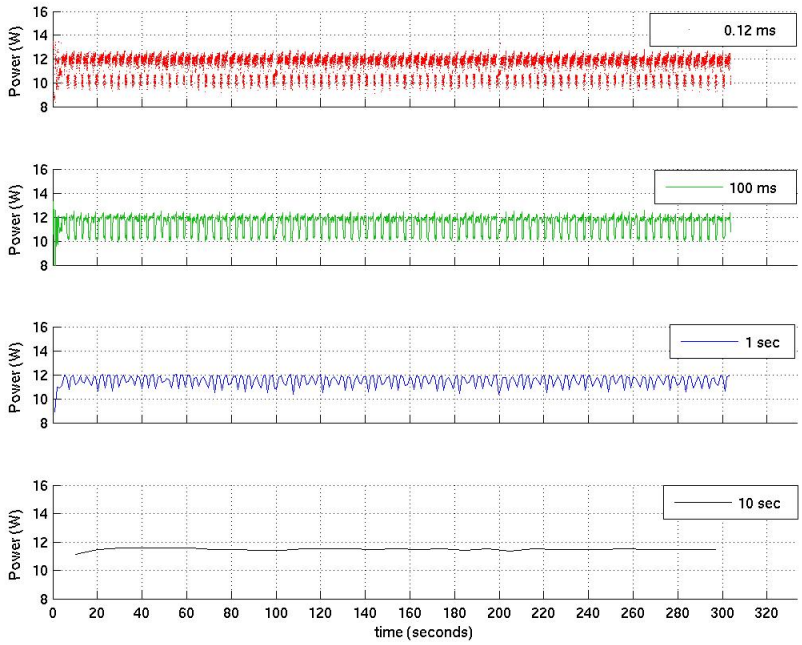


Figure 4.4: Average power per sample at multiple sample-interval lengths, from 12 ms per sample to 10 seconds per sample, for benchmark `ammp`.

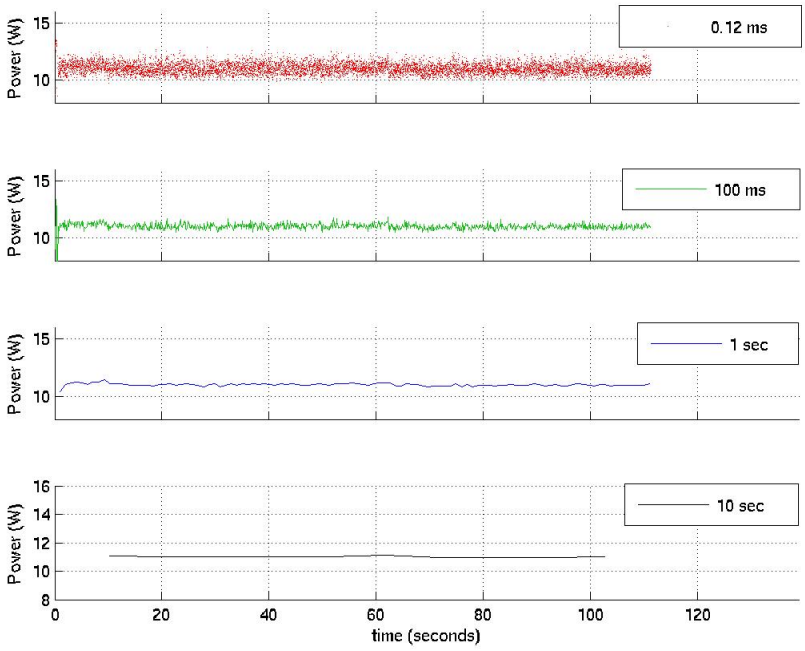


Figure 4.5: Average power per sample at multiple sample-interval lengths, from 12 ms per sample to 10 seconds per sample, for benchmark `art`.

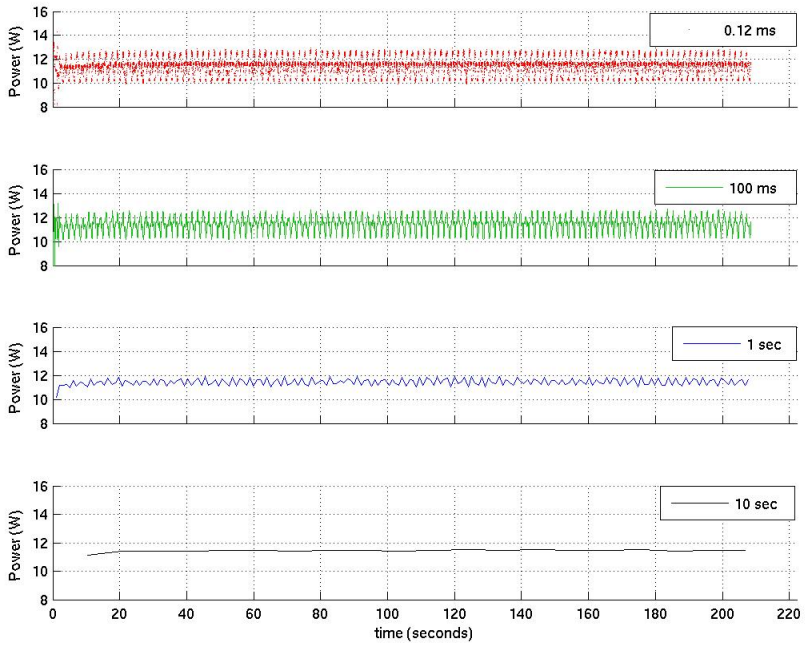


Figure 4.6: Average power per sample at multiple sample-interval lengths, from 12 ms per sample to 10 seconds per sample, for benchmark `lucas`.

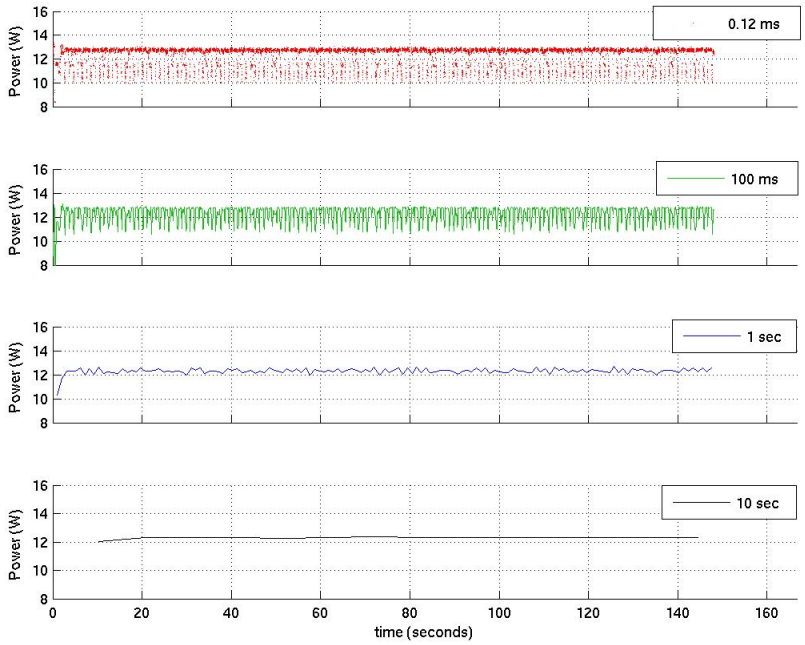
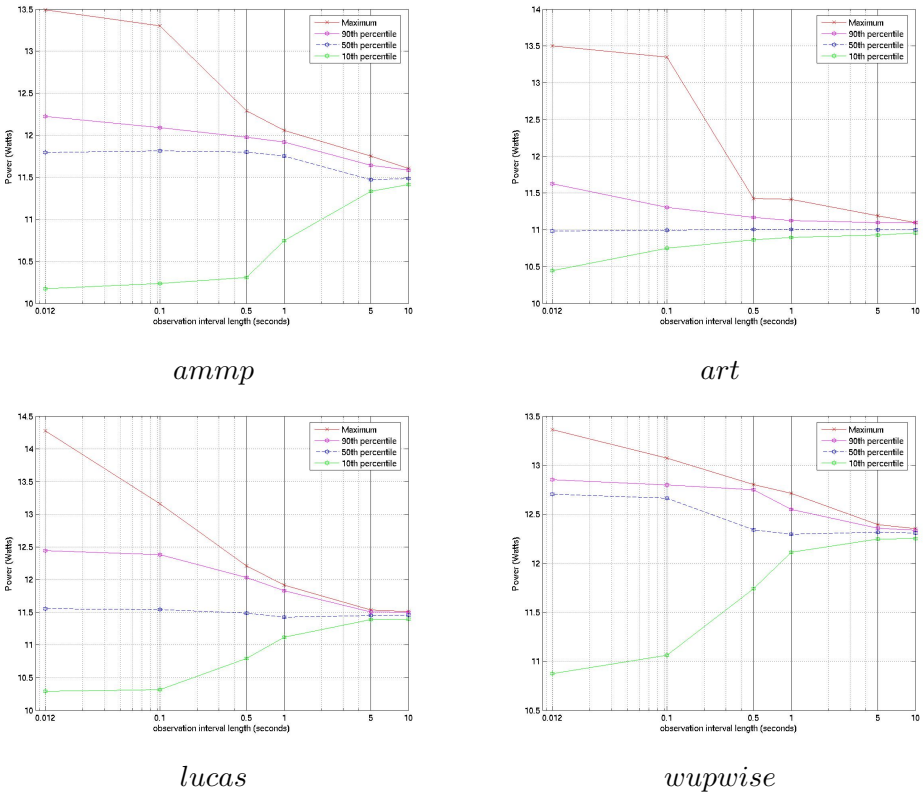


Figure 4.7: Average power per sample at multiple sample-interval lengths, from 12 ms per sample to 10 seconds per sample, for benchmark `wupwise`.



ammp *art*

lucas *wupwise*

Figure 4.8: Workload sensitivity to measurement interval length.

Median Sensitivity	Maximum Value Trends	
	Sharp	Smooth
Steady	`art`	`lucas`
Variable	`ammp`	`wupwise`

Table 4.5: Interval length sensitivity taxonomy with example benchmarks.

for 10^{th} percentile, 50^{th} percentile, 90^{th} percentile, and the maximum power value. The 10^{th} percentile point indicates that 10% of all power values are equal to or below that power value. The 50^{th} percentile point is the median, the power value in which half of the samples are above and half are below; note that the median may have a different value than the mean value. The 90^{th} percentile point indicates that 90% of all power values are equal to or below that power value.

We observe two categories of workload sensitivity to measurement interval length: maximum-value trends and median variability. Table 4.5 summarizes the taxonomy of sampling-interval sensitivity characteristics with examples from SPEC CPU2000 benchmarks.

First, the maximum value observed depends upon the granularity of measurement interval, as long sampling intervals obscure visibility of short power spikes. Benchmarks that exhibit occasional extreme power values exhibit a large power delta between the maximum and 90^{th} percentile values. The sampling sizes at which outlier values are visible varies by benchmark.

The `art` benchmark shows a sharp drop in the maximum-90% delta, from about 2 Watts at 0.1-second intervals to less than a half Watt at 0.5-second intervals. `Ammp` exhibits a 1-Watt drop at the same time scales. In `art` and `ammp`, the

sudden change in maximum values indicates the points at which outlier values no longer dominate the sampling window. Conversely, maximum values for `lucas` and `wupwise` form a smooth curve across the sampling-interval range.

Maximum value trends have implications for power control. Enforcing a fixed power budget may lead to different p-state choices at different sampling intervals, as the maximum observed value varies for the same benchmark. For example, the maximum power recorded for `art` varies by about 20% between the 10-ms and 10-second sampling intervals. An important consideration for power control is the system components' tolerance for power overshoots. Voltage regulators, for example, may detect power violation within 10 ms and enact a drastic over-current protection mechanism [17] that a fine-grain controller could more gracefully handle.

Second, the median (50^{th} percentile) point may be steady or may vary across sampling interval sizes. The medians of benchmarks `art` and `lucas` are insensitive to sampling interval length for different reasons: `art` exhibits steady behavior at all sampling sizes, while `lucas` contains a balance of high and low points that contribute equally to the median. The median levels vary with sampling interval size for benchmarks `ammp` and `wupwise`, which both exhibit a power distribution at 10-15 ms of most points clustered in a typical power range, with lower-power points scattered throughout the benchmark. The median fluctuates with sample size, depending on whether the lower-power execution is captured as independent samples with small interval lengths, or are averaged with more typical points at longer interval sizes.

Variable medians indicate that data observed at one interval length would not be representative at other sampling intervals, an important consideration for

interpolating or extrapolating data. For example, phases or other patterns detected with 50 ms sampling may not be apparent at 5-second intervals.

4.5.3 Effects of Sampling Interval Length

Longer intervals attenuate power fluctuations, which has two important ramifications. First, steadier behavior at longer sampling intervals could improve control stability and power prediction accuracy. Second, longer sampling intervals can obscure power-limit violations on short timescales. The system may tolerate overshoots with an acceptable average for some considerations, such as total energy consumption targets, but for other considerations, such as total current draw from a power supply, overshoots may be a serious concern that warrant shorter monitoring time scales.

It would be possible to vary the monitoring time interval to tailor the observation to the granularity necessary to ensure adequate control. For example, power could be monitored at a 1-second interval in the typical case of redundant power supplies, and switch to 10 ms (or less) intervals under special circumstances that would require tighter control, such as a power-supply failure in one of the redundant units.

Chapter 5

Response to P-State Management

Commercial computing systems employ several power management techniques. We designed experiments to observe the time scale and magnitude of response to p-states, and to identify considerations beyond p-states that would affect the outcome of power and thermal management.

In this chapter, we characterize the response to two power management techniques, dynamic voltage and frequency scaling (DVFS) and clock throttling on a commercially available system, a Pentium M platform, to capture realistic power and thermal responses. The characterization measurements in this chapter form a baseline of expected behavior for power and thermal responses to p-state changes, which we use to develop our PET manager prototype.

We execute microbenchmarks from the `MS-Loops` suite, designed to capture both core and memory subsystem activity with floating-point benchmarks with varying data footprint sizes. For each microbenchmark, we record power and performance for three dataset sizes corresponding to core-bound, intermediate, and memory-bound programs. We observe that for techniques applied individually, DVFS is more effective than clock throttling at reducing power while preserving performance.

We also apply both techniques together in each of the 64 combinations of 8 frequencies and 8 clock throttling levels. We demonstrate that a performance or power target can be achieved with multiple combinations of DVFS and clock throttling settings; however, DVFS provides superior power-performance efficiency.

We follow the power and performance analysis with an investigation of the thermal response to DVFS. We observe the transient and steady-state responses to each DVFS state with custom microbenchmarks. We also record the full SPEC CPU2000 benchmark suite executing at maximum speed to determine the effects of a realistic workload with fluctuating power consumption. Then, we leverage insight of the power response to p-states to understand and predict the CPU temperature response to p-states, and identify the contributions of external influences of air flow and ambient temperature on CPU temperature.

The recorded temperatures in the characterization study are unique to the specific Pentium M system; we expect that the characterization methodology and observed trends, such as the time constant for transient response, are applicable to other platforms, as well. Measured data from live hardware illustrate the complex interaction between power and temperature and provide insight for future work in power and thermal management by identifying the magnitude and timescale of responses to power-management settings and cooling capacity.

We apply characterization data to build predictive models of performance, power, and temperature for the Pentium M system. We use measured data for present conditions to predict what the outcome would be for all other p-state choices, presenting the PET manager an informed choice for p-state management.

5.1 Power and Performance Characterization

Current and next-generation commercial computing systems have access to several power management techniques, and to employ them effectively, we need to understand the effects on performance, power and temperature. We characterized application response to two power management techniques, DVFS and clock throttling, applied individually and in combination on a Pentium M system with a set of microbenchmarks designed to capture core and memory subsystem activity. We analyzed microbenchmark power and performance for three dataset sizes corresponding to core-bound, intermediate, and memory-bound programs. We found that for techniques applied individually, DVFS is more effective than clock throttling at reducing power while preserving performance. We also applied both techniques together in each of the 64 combinations of 8 frequencies and 8 clock throttling levels. We demonstrated that a performance or power target can be achieved with multiple combinations of DVFS and clock throttling settings, yet DVFS provides superior power-performance efficiency, especially with memory-bound applications.

5.1.1 P-state Comparison

Appendix A contains details of DVFS and clock throttling mechanisms applied independently to microbenchmark execution. This section summarizes the p-state response to each mechanism. Figures 5.1 through 5.3 present normalized data for one microbenchmark, `DAXPY` executing with 3 footprint sizes: L1, L2, and main memory for each p-state. The two types of p-states are applied separately: DVFS points are unthrottled and clock throttling points are at the maximum 2 GHz

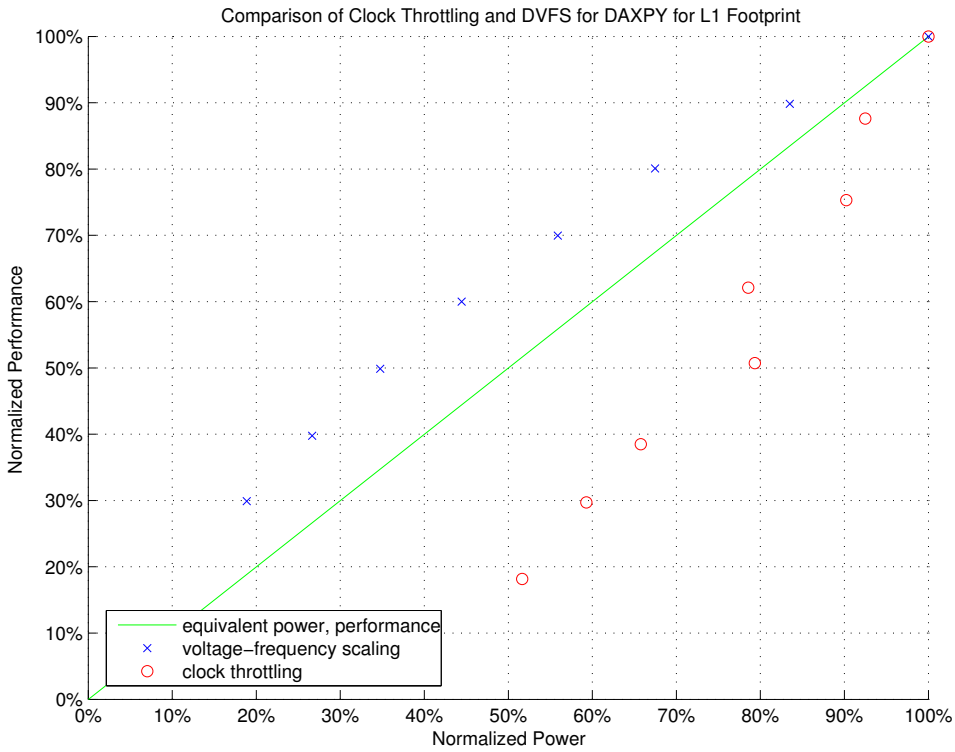


Figure 5.1: Comparison of power and performance response to clock throttling and DVFS for L1-resident benchmarks.

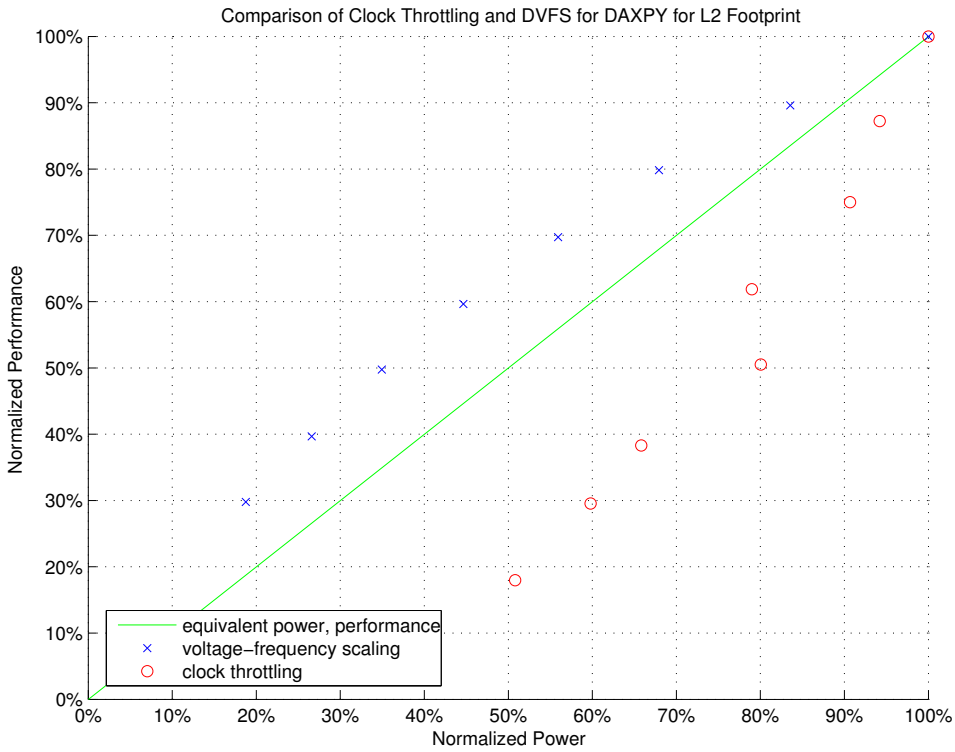


Figure 5.2: Comparison of power and performance response to clock throttling and DVFS for L2-resident benchmarks.

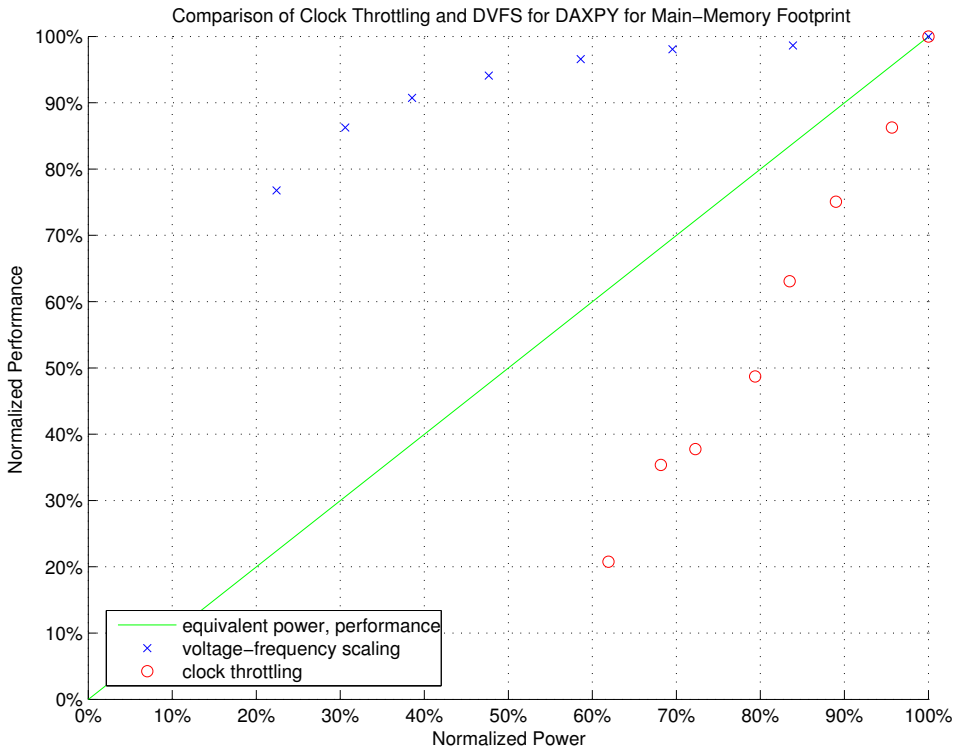


Figure 5.3: Comparison of power and performance response to clock throttling and DVFS for non-cache-resident benchmarks.

frequency. In these charts, we exclude static and frequency-independent power. The remaining power, that which is controlled by the DVFS and clock throttling mechanisms, is normalized to the maximum observed power at the unthrottled 2 GHz p-state. Program execution latencies are normalized to the maximum performance (minimum latency) point, also at unthrottled 2 GHz. The diagonal line indicates the break-even point where performance and variable power are affected equally by a technique, for example, 50% of maximum variable power and 50% of maximum performance. Above the line are better power-performance efficiency points; below the line, performance degradation is worse than power savings.

Clock throttling results in greater performance degradation per power reduction, with data points below the break-even point for all benchmarks and footprint sizes. DVFS reduces power with less impact on performance than clock throttling. The power efficiency is especially pronounced in memory-bound workloads, which benefit from fewer core clock cycles stalled for memory accesses. In contrast, lower effective frequencies with clock throttling do not benefit from fewer core cycles, as evident in Figure 5.3, where the maximum clock throttling consumes approximately 60% of the maximum power but yields only 20% of best-possible performance, compared to DVFS which reduces power to 20% of maximum yet maintains nearly 75% of performance.

5.1.2 Combined Mechanisms: Clock Throttling and DVFS

We measured performance and power with each microbenchmark at each of the 64 p-state combinations (eight DVFS and eight clock throttling levels) and

for each of three data working set footprints. Figure 5.4 reports the maximum measured power and total execution latency for the SPEC CPU2000 benchmark `gap` at each DVFS-throttle combination. The response to p-state combination reflects the individual clock throttling and DVFS characteristics, where DVFS provides superior power efficiency. High frequency, heavily throttled options such as the 2 GHz/throttle 1 point produce longer latencies and higher power (worse in both metrics) than low-frequency unthrottled options, such as 600 MHz/unthrottled.

The figure illustrates that multiple p-states could meet either a performance or power requirement. A power manager will need to choose from a rich selection of options, many of which are sub-optimal. Consider the point labeled 600 MHz-throttle 1, near 1800 seconds of execution time and 2 Watts of power consumption. This p-state combination provides the lowest possible power consumption for this system, with severe performance degradation compared to other points with near-minimum power consumption and half the execution time. In general, simply minimizing power will not yield the most desirable response. A power management approach must discern the best choice among many options according to explicit preferences, providing the ability to provide the absolute minimum power during an emergency or more performance-aware choices during typical operation.

We illustrate the effects of combined DVFS and clock throttling with the `DAXPY` benchmark for three levels of memory boundedness: L1-resident, L2-resident, and off-chip memory. Graphs in Figure 5.5 compare the trends for performance and power for each combination of DVFS and clock throttling. The grid intersection points indicate combinations of DVFS and clock throttling combinations, such as

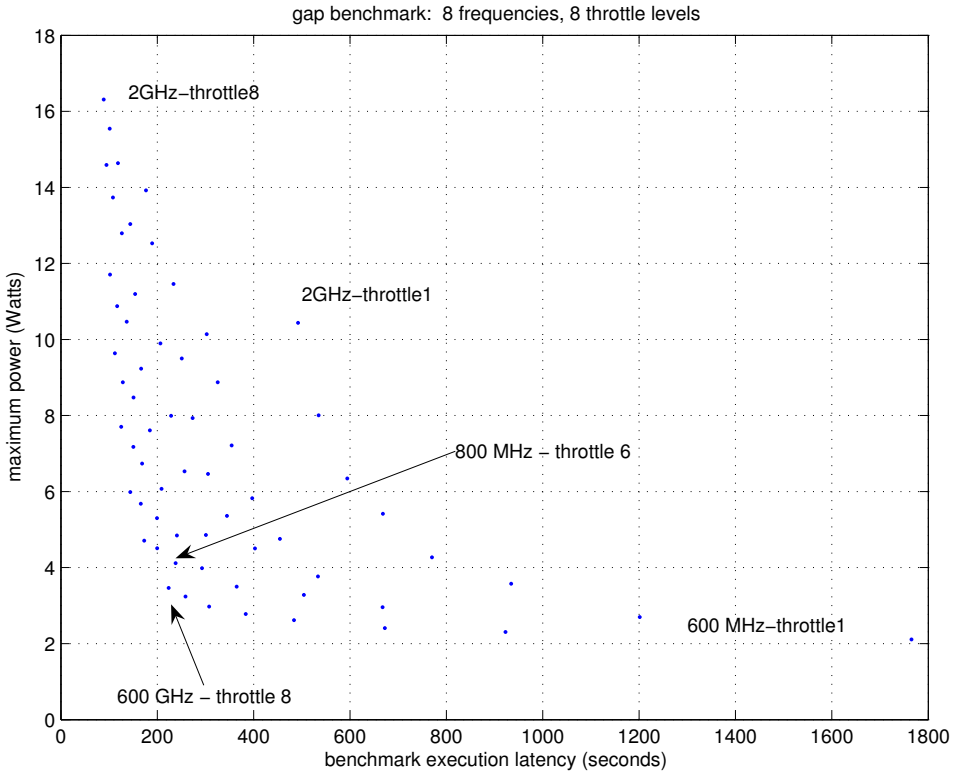


Figure 5.4: Maximum power and total execution time for DVFS and clock throttling p-states applied to `gap` benchmark execution.

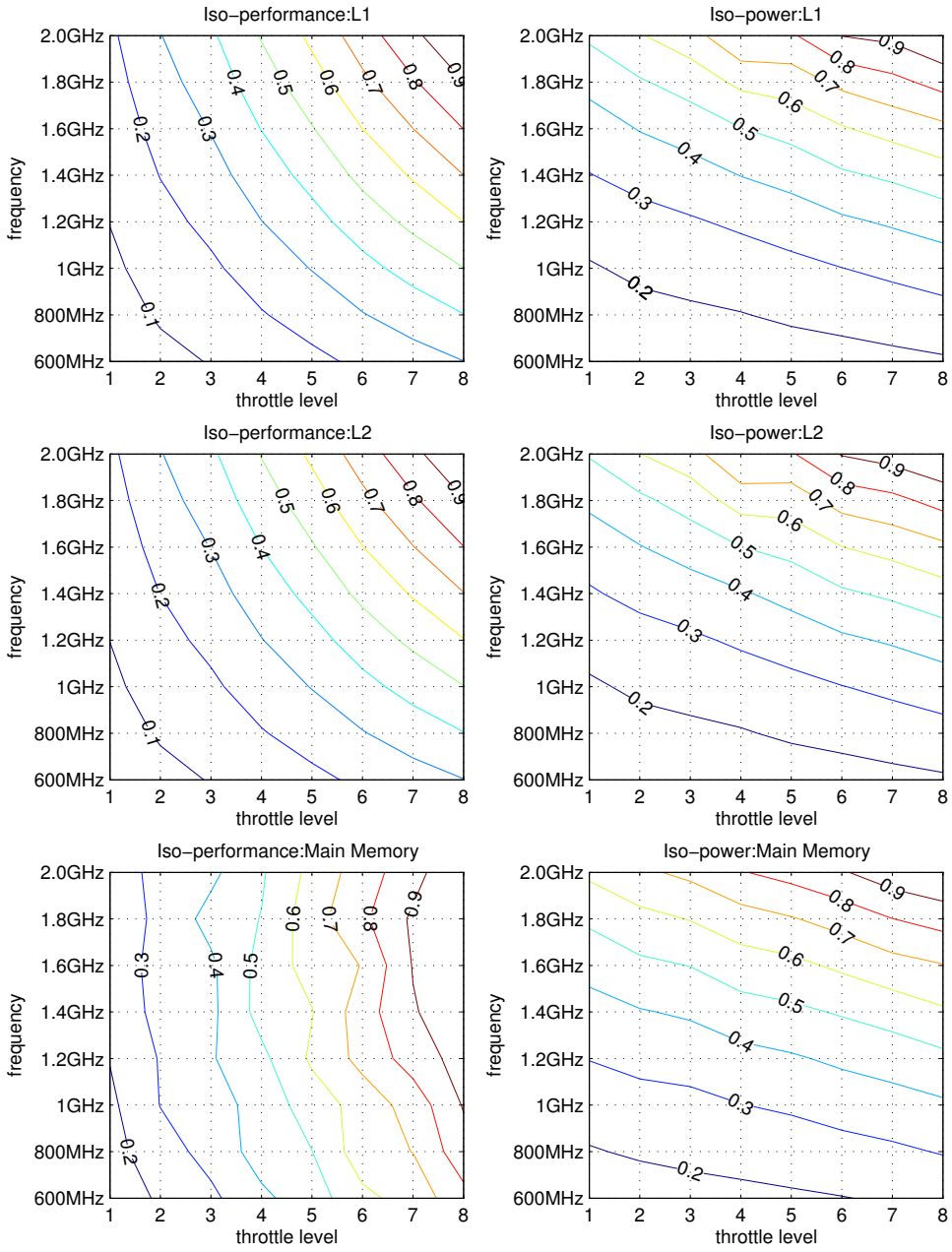


Figure 5.5: Microbenchmark performance and power isolines for `DAXPY`.

1 GHz paired with throttle level 6, or 600 MHz paired with throttle level 2. Isolines indicate the percentage of maximum power or performance that the system achieves, interpolated between discrete p-state settings to show trends. The isoline graphs illustrate two important lessons we observed in the characterization study. First, a performance *or* power target can be achieved with different p-state settings. For example, the compute-bound L1-resident `DAXPY` benchmark reaches a target of 50% of maximum performance with either clock throttle level 4 paired with a frequency of 2 GHz, or with 1 GHz paired with throttle level 8 (unthrottled).

Second, the effect of DVFS p-states varies with the degree of memory dependence, while clock throttling does not. Thus, combinations of DVFS and clock throttling produce different results depending upon memory access characteristics, even for workloads with the same instruction mix. A microbenchmark with a main-memory sized working set, too large to fit in the on-chip caches, creates a memory-bound workload that spends a large portion of execution time waiting for data from off-chip memory. For the memory-bound version of `DAXPY`, the 2 GHz/throttle level 4 combination still produces 50% of maximum performance, but the unthrottled 1 GHz setting achieves much higher performance, 90% of maximum performance. Memory-bound applications can benefit from substantial power reduction at lower frequencies and voltages with much less effect on performance, which favors DVFS rather than clock throttling. For example, to achieve 50% of maximum performance, a memory-bound workload at 2 GHz and throttle level of 4 requires approximately 80% of maximum power. Alternatively, a lower frequency of 800 MHz and throttle level 5 yields similar performance, and requires only 25% of maximum power.

We investigated clock throttling to explore tradeoffs between combinations of p-state mechanisms. The range of power control varies with each mechanism, and DVFS is sensitive to workload behavior whereas clock throttling is not, thus offering a wide range of responses with two mechanisms. We anticipated that the combination of clock throttling would provide a benefit for performance and power management by providing a finer granularity of control, 64 options with two mechanisms rather than only 8 with a single mechanism.

Instead, we found clock throttling to be significantly less power efficient than DVFS across the board. Two situations not considered in these experiments could warrant the use of clock throttling for power and performance management: (a) a quick change to a clock throttling p-state could be a good control choice for a fast response without incurring a stall (up to 500 μs on this system), and (b) clock throttling in conjunction with the lowest-frequency DVFS p-state can reduce power to the bare minimum.

In our experimental infrastructure, the overheads for changing voltage and frequency are negligible at the millisecond scale, and we typically do not encounter situations that require power below 3 Watts. Thus, we will focus our attention on the most effective knob, DVFS, for further study.

5.2 Power Influence on CPU Temperature

The system's thermal response is determined by two factors in opposition: heat generated by power dissipation, and heat conducted by the cooling system. In this section, we observe the effects of power on temperature under maximum

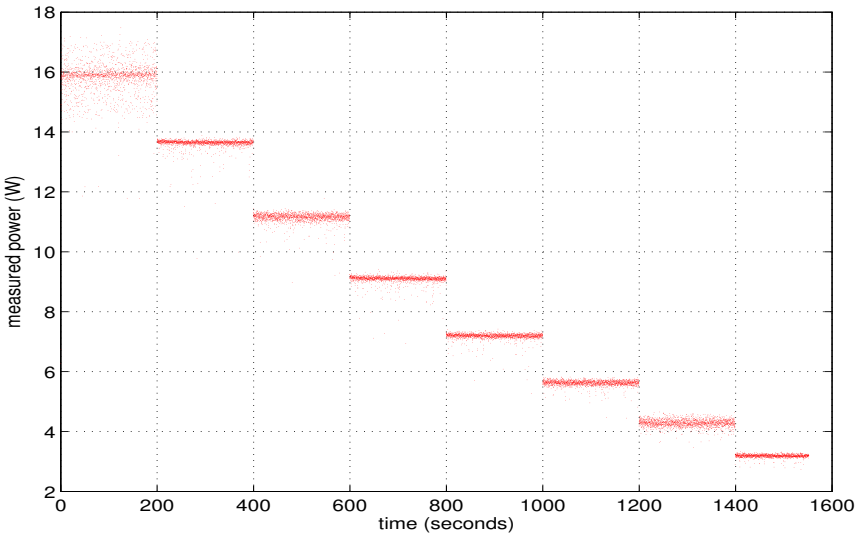


Figure 5.6: Power for the `daxpy` microbenchmark at each of 8 p-states (decimated data).

cooling conditions and the effects of the cooling environment on CPU temperature under steady-power conditions, with the steady-behavior microbenchmarks of the `MS-Loops` suite.

5.2.1 Transient Response

Figure 5.6 shows a continuous trace of the `daxpy` benchmark executing as the p-state changes every 200 seconds, starting at the maximum frequency-voltage pair at 2 GHz, descending in 200 MHz steps to 600 MHz. In this experiment, the monitoring software recorded power samples at a rate of 20 samples per second (50 ms each), with one temperature sample for every 2 power samples (100 ms between unique samples). Power exhibits a clear relationship with DVFS setting; each step in frequency is accompanied by a sharp drop in power.

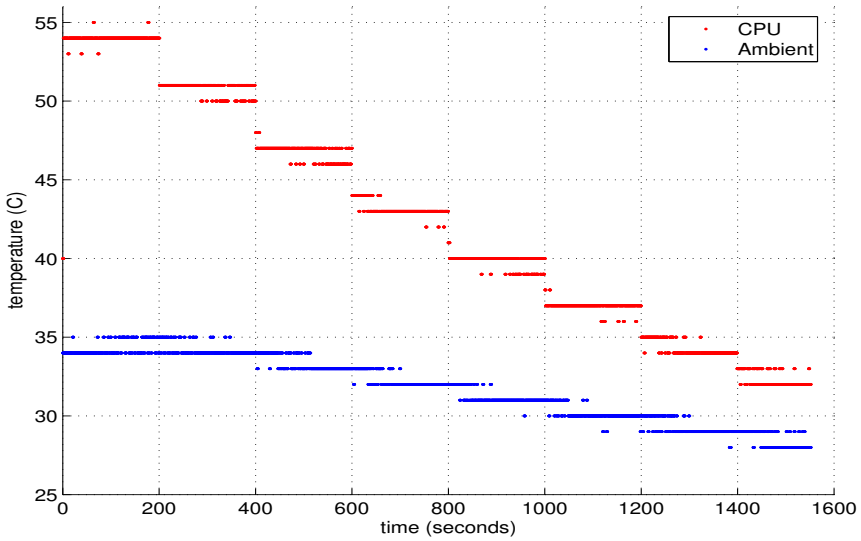


Figure 5.7: CPU temperature for the `daxpy` microbenchmark at each of 8 p-states (decimated data).

Figure 5.7 shows the corresponding trace of measured temperatures for the `daxpy` microbenchmark executing during descending frequency steps. Measured temperature bounces between integer values on the graph due to the coarse 1 °C resolution of the measurement. We observe that each p-state step to a lower frequency causes an initial sharp drop in CPU temperature within the first 50 ms sample, then an additional descent of 1-3 degrees over several samples in most cases, followed by a longer period of slight decrease in temperature over the span of several seconds. Figure 5.8 shows two frequency changes, from 1800 MHz to 1600 MHz to 1400 MHz. After the change to 1600 MHz at the 400-second mark, the CPU temperature settles to 47 °C within approximately 20 seconds and remains stable for about one minute, then begins a gradual, non-monotonic decrease of about 1 degree. At the 600-second mark, the p-state changes to 1400 MHz, at which point the temperature drops to

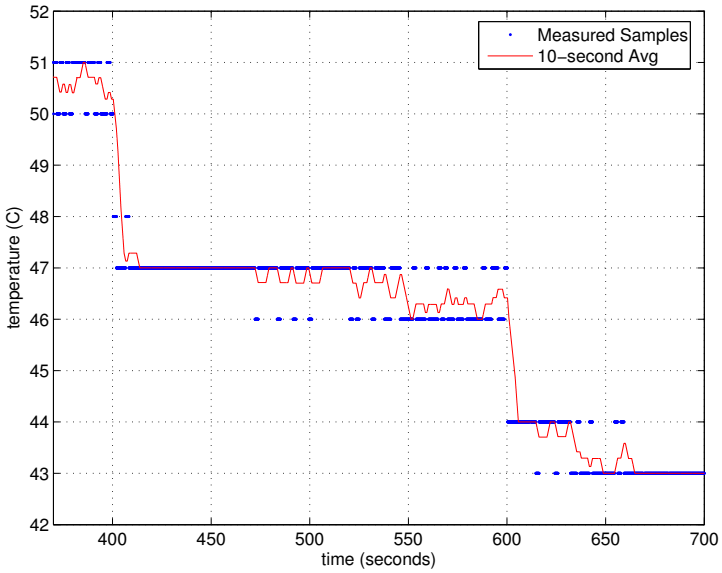


Figure 5.8: CPU temperature for frequency transitions 1800 MHz to 1600 MHz to 1400 MHz.

44 degrees, then settles to 43 °C after approximately one minute.

In most cases, the CPU temperature dropped by 3 degrees following the p-state transition, typically with an immediate change of 2-3 degrees and another degree gradually throughout one minute after a p-state change. Then, the influence of gradually changing ambient temperature is noticeable as the CPU temperature continues to cool for another 1 °C over the next few minutes. Figure 5.7 shows both CPU and ambient temperatures decreasing over time. As the CPU temperature decreases, the ambient also decreases, which reduces the thermal load on the heatsink, which allows the CPU temperature to further decrease.

At the low end of the frequency spectrum, when power is lowest and the cooling system is most capable of removing the CPU-generated heat from the system,

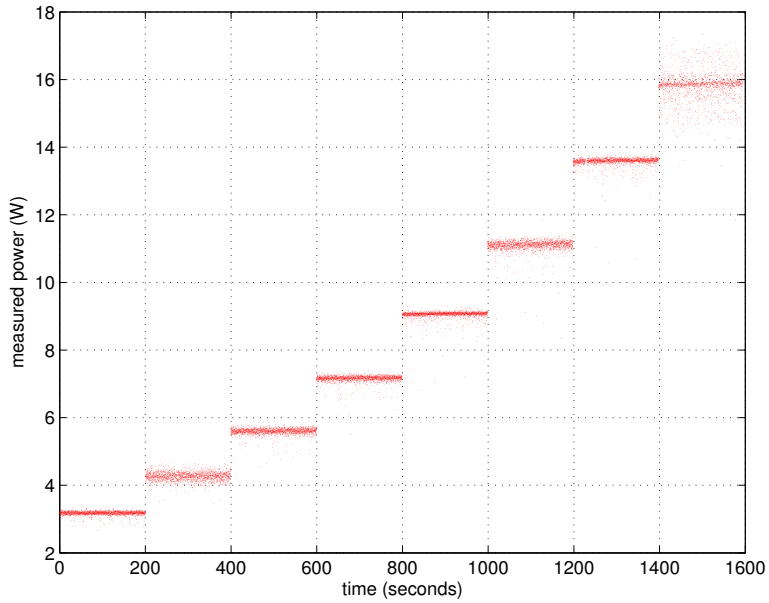


Figure 5.9: Power for the `daxpy` microbenchmark, recorded with ascending frequencies (decimated data).

the temperature decreased by a total of 4 °C after the transition from 800 MHz to 600 MHz.

Figures 5.9 and 5.10 show the complementary case of the `daxpy` benchmark and p-state steps with ascending frequencies. The mean power for each p-state is the same for ascending and descending frequencies, within 70 mW. Temperature is more variable, as it depends on the room conditions, including ambient temperature and airflow. The mean temperatures varied between the ascending and descending cases by up to 1.5 degrees. Like the descending case, measured power values form bands of steady behavior for the `daxpy` benchmark, with thick bands for 800 MHz and 1600 MHz, and a wider, more sparse band for 2 GHz in Figure 5.9. Figure 5.10 shows how ambient and CPU temperature increase as the p-state ascends through

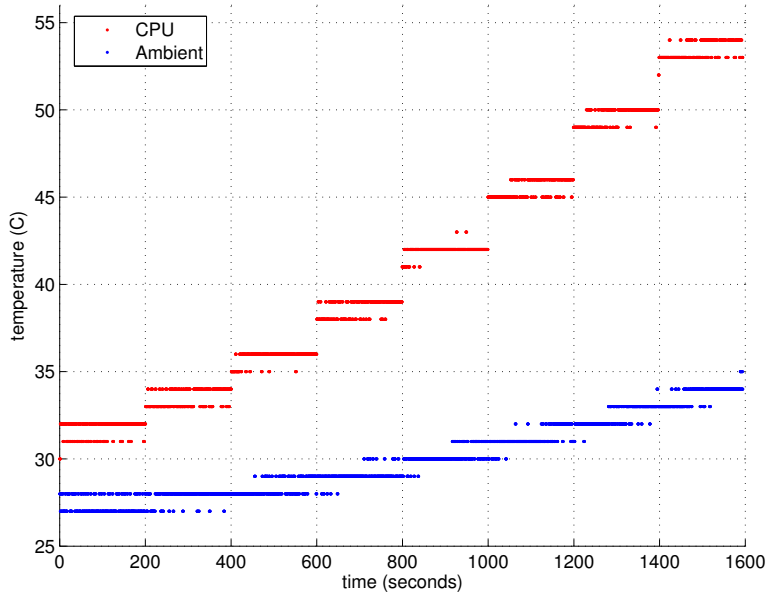


Figure 5.10: CPU temperature for the `daxpy` microbenchmark, recorded with ascending frequencies (decimated data).

all frequencies.

Figure 5.11 shows the timescale for ascending p-state transitions from 800 MHz to 1 GHz to 1.2 GHz, displaying individual measured points and a continuous line for the 10-second moving average. Like the descending case, the temperature initially changes a small amount quickly (1-3 °C), then continues for another degree over about one minute. Figure 5.11 does not show a second-stage heating effect from the ambient in this portion of the trace; it is possible that the ambient influence created a temperature rise less than 1 °C, and thus is not measured with the thermal sensor's 1-degree resolution. As Figure 5.10 shows, the ambient does rise through time as the CPU temperature rises.

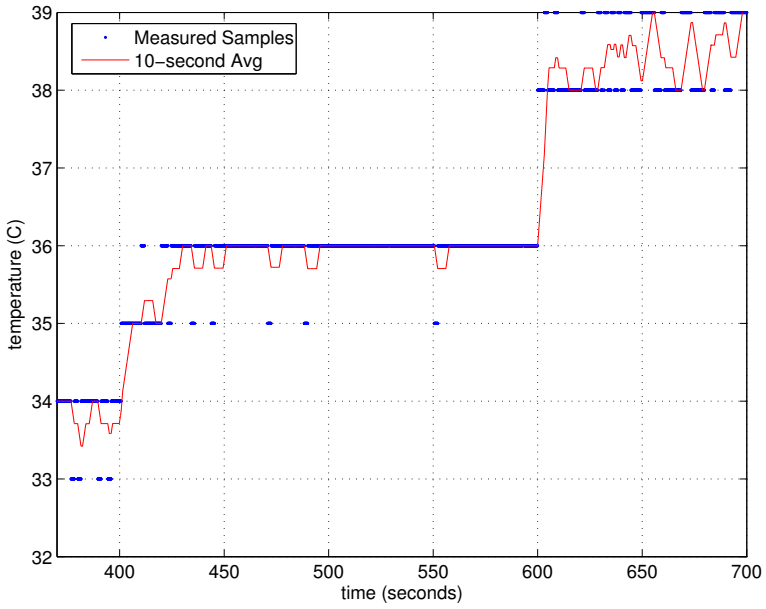


Figure 5.11: CPU temperature for frequency transitions 800 MHz to 1000 MHz to 1200 MHz.

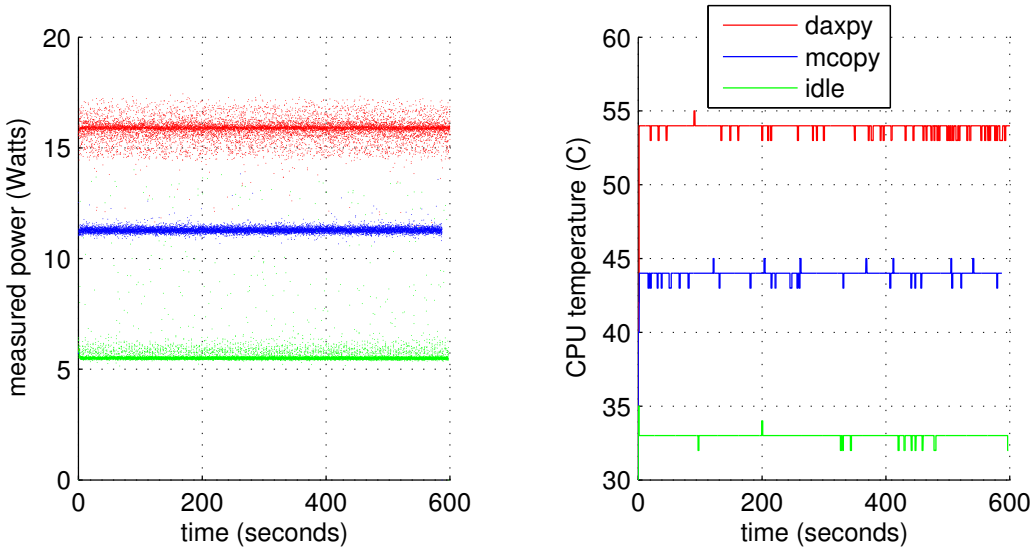


Figure 5.12: Microbenchmarks' steady-state power at 2 GHz.

5.2.2 Steady-State Response

Figure 5.12 shows measured power and CPU temperatures for a series of 3 microbenchmarks. The benchmarks each executed twice consecutively in the order of `mcopy` twice, `daxpy` twice, and `idle` twice. The figure shows only the second execution of each benchmark, which captures the steady-state behavior. Each benchmark executed for approximately 10 minutes at a frequency of 2 GHz. In Figure 5.12, note that each benchmark maintains a steady band of power dissipation throughout its execution, and the power consumption varies by benchmark, even at the same p-state. Slight differences in actual temperature are amplified as values are rounded to the nearest full degree, resulting in measured CPU temperatures varying by +/- 1 °C for steady-state behavior.

The standard deviation in power and thermal measurements are small, less

than 1 Watt and 1 °C, indicating that the steady-state measurements are indeed steady. `Mcopy` and `daxpy` have well-defined continuous behavior and execute at a high priority, resulting in low standard deviation in measured power. The spread in measured temperatures is slightly higher. The coarse resolution of the temperature measurement magnifies small differences between integer values with rounding error, and the true temperature may actually have larger fluctuations due to high heat output relative to the cooling system capability.

The `idle` microbenchmark, on the other hand, captures variable, uncontrolled background tasks executing during the `sleep` command, not a true idle state, and is thus less steady in workload behavior and recorded power throughout the range of p-states. However, the temperature spread is lower than the other benchmarks. We theorize that the smaller temperature range is due to two factors; a steady-state temperature near an integer value (fewer fluctuations due to alternating round-up and round-down errors than) and also possibly that the temperature is steadier due to lower heat output from the CPU, which would be more easily removed by the cooling system. With the maximum fan speed, the lower power of `idle` is most easily managed by the cooling system; higher-power cases such as `daxpy` at high frequencies shows more thermal variation.

A fairly linear relationship between power and temperature is evident in Figure 5.13. The slightly different slopes for `daxpy` and `mcopy` points in Figure 5.13 are most likely due to temperature sensor placement relative to workload-specific hotspots on the processor. The single sensor may be closer to `daxpy`'s hotspots than for `mcopy`. Additional sensors on-die would give a more complete picture of

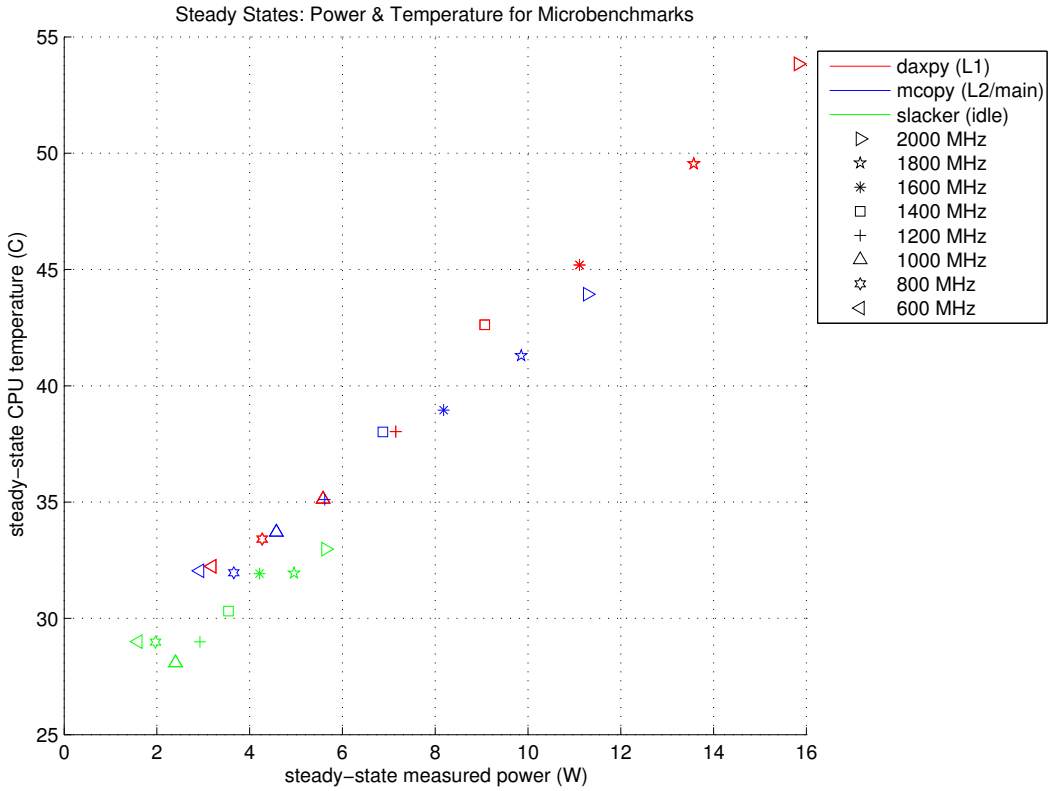


Figure 5.13: Mean power vs mean temperature for 3 microbenchmarks at 8 DVFS p-states.

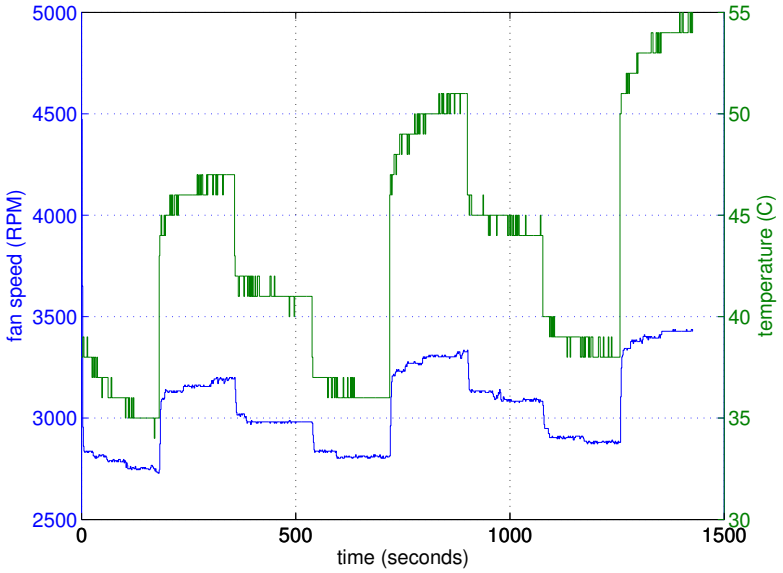


Figure 5.14: P-state changes in 400 MHz steps, with temperature-tracking fan.

the power-thermal relationship, yet even a single measurement point provides an indication of expected behavior, within a few degrees Celsius.

5.3 Cooling System Influence on Temperature

In this section, we analyze the effect of the CPU fan on temperature. The default behavior for the CPU fan is to track CPU temperature within a specified temperature range, increasing fan speed revolutions per minute (RPM) in response to rising temperatures and decreasing with hysteresis to follow falling temperatures. Above the specified maximum temperature, the fan runs continuously at maximum speed, around 4500 RPM. Below the specified range, the fan speed is set to a user-defined minimum speed, which may be set to zero RPM to disable the fan.

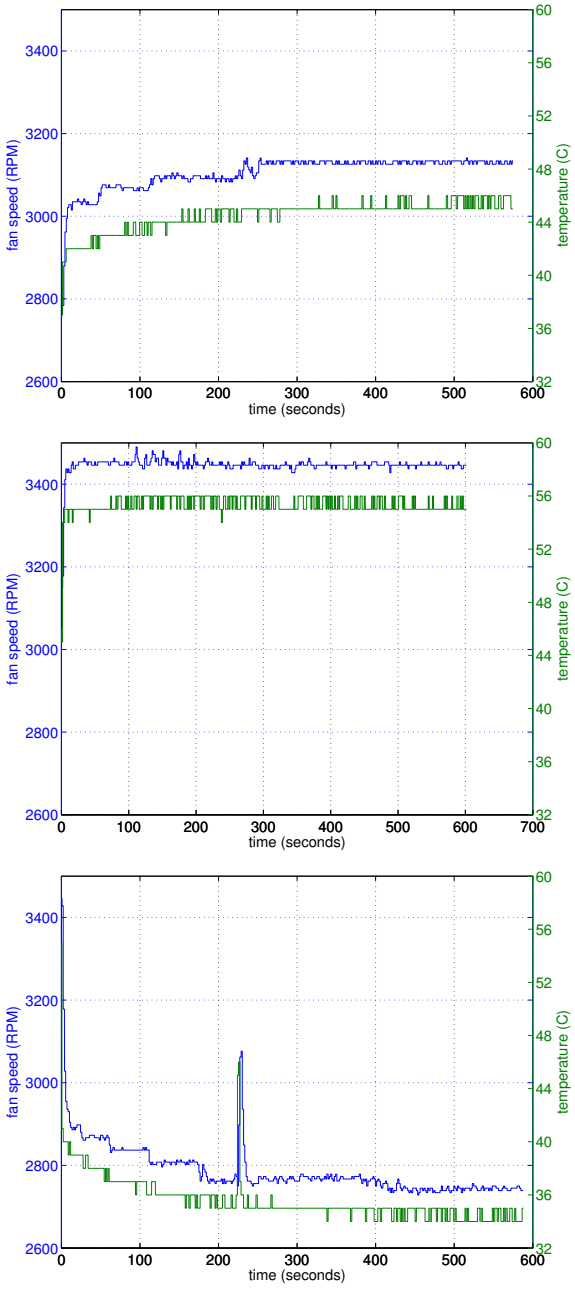


Figure 5.15: Temperature-tracking fan: CPU temperature (green) and fan speed (blue). Due to different power levels, `mcopy` warms relatively slowly, `daxpy` warms quickly, and the system cools during the `idle`. The thermal spike in the `idle` test is the result of a short spurt of processor activity.

While turning the fan off is an extreme case, it demonstrates system properties under high-temperature conditions, such as those in the hot zones within server rooms or other under-cooled situations. Fans and blowers consume a large portion of server power budgets [57]. Understanding the effects of disabling fans and modulating fan speed are a first step toward managing overall system power by trading cooling capacity for power consumption.

5.3.1 Temperature-tracking Fan

Figure 5.14 shows the CPU temperature and fan speed as the p-state changes with large frequency differentials, increasing in 1 GHz steps and decreasing in 400 MHz steps for the `daxpy` benchmark. We observe similar thermal behavior as the previous transient response under maximum-fan conditions, with an initial sharp drop or rise, followed by a longer tail of temperature change.

Figure 5.15 shows the CPU temperature and fan speed for three microbenchmarks executing at 2 GHz while the fan operates under typical temperature-tracking conditions. In this experiment, the benchmarks executed in the order of `mcopy`, `daxpy`, `idle`, at 2 GHz. The first benchmark `mcopy` warms gradually to 45-46 °C, `daxpy` quickly reaches a steady level of 55-56 °C, and `idle` cools to 36 °C due to its lower power consumption. The time scale to reach steady-state temperatures varies from about 2 minutes for high-powered `daxpy`, about 5 minutes for `mcopy` and `idle`. The fan speeds reflect the temperature trends, with higher fan speeds for the hotter benchmarks. A short spike of activity during the `idle` test caused a temperature jump, followed by an increase in fan speed.

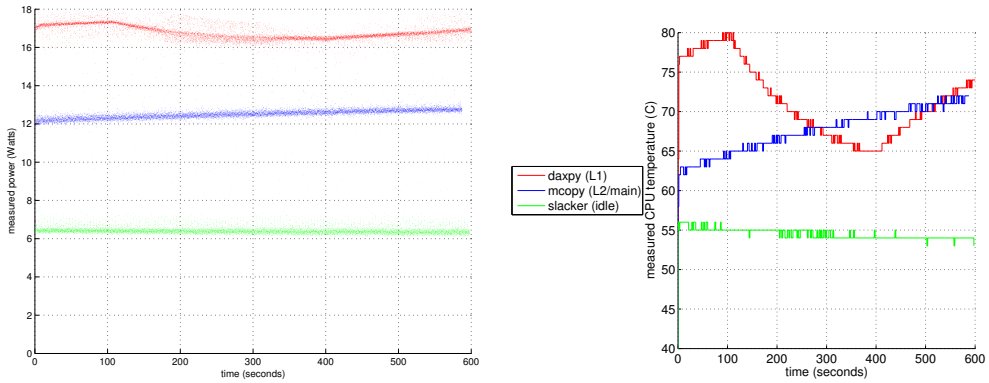


Figure 5.16: Disabled fan: CPU power and temperature for 3 microbenchmarks at 2 GHz.

5.3.2 Disabled Fan

We designed experiments with the CPU fan disabled to capture the Pentium M's response for steady-state and transient power and thermal response for harsh, under-cooled situations. Figure 5.16 show the power and thermal measurements for each of the 3 benchmarks while the fan is disabled.

We disable the fan indirectly by setting T_{min}, the temperature at which the fan turns on, to 80 °C. In most cases, the Pentium M CPU operates well below 80 °C and the fan remains off. However, if the CPU temperature does reach T_{min}, the fan turns on, as evident in the `daxpy` case. Power and temperature for `daxpy` rise, dip, and rise again due to the fan operation. `Daxpy`'s high power consumption leads to high temperatures while the fan is disabled, quickly reaching the 80 °C threshold for the fan to engage. The fan continues to spin at a low rate until the temperature decreases below a lower threshold, empirically observed to be 65 °C in these experiments, which is 15 °C below the upper threshold. The lower-powered

benchmarks `mcopy` and `idle` do not exceed the 80 °C limit and thus the fan remains disabled throughout their execution. One interesting effect of disabling the fan is the difference between the `mcopy` and `idle` benchmarks. The CPU temperature rises for `mcopy` yet decreases slightly for `idle`; power rises for `mcopy` in concert with the rising temperature, and remains steady for `idle`.

Leakage current is exponentially dependent on temperature; higher temperatures produce higher leakage current and greater power consumption. Power and temperature can exhibit a feedback effect of increasing temperatures raising leakage current, in turn increasing power consumption, which generates more heat and further raises the temperature. Figure 5.17 shows the effect in detail for the p-state change from 800 MHz to 1800 MHz for the `daxpy` benchmark with the fan disabled. Both temperature and power are steady at 800 MHz, and after the p-state transition, both temperature and power continue increase.

The thermal runaway feedback effect is more pronounced at power levels above 10 Watts in this system. Figure 5.18 shows an experiment with stepped p-state levels for the `daxpy` benchmark and disabled fan. Initial temperatures for 1800 MHz and 1400 MHz p-states are similar, yet temperature rises much more quickly for the higher-power 1800 MHz state. At 1600 MHz, both power and temperature rise quickly despite lower T_{CPU} compared to the 1400 MHz state, which has a slower temperature rise and negligible power increase. We expect that the system is better able to dissipate the extra heat generated from leakage current during lower power levels (lower total heat output), reducing the effect of leakage power on T_{CPU} and thus attenuating the feedback effect.

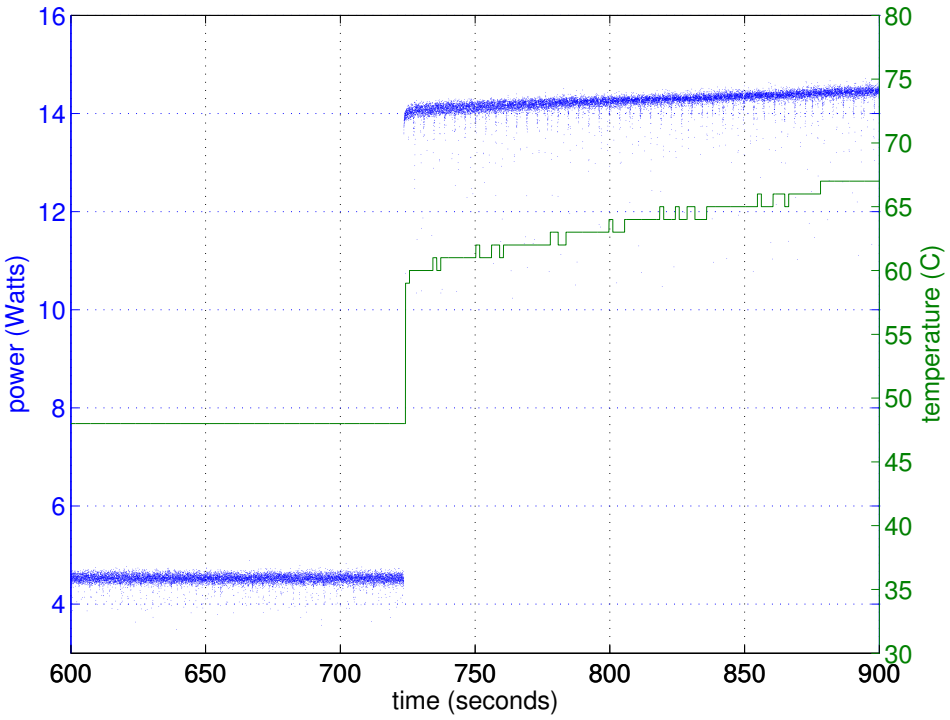


Figure 5.17: Disabled fan: p-state change from 800 MHz (steady) to 1800 MHz (thermal-power interaction).

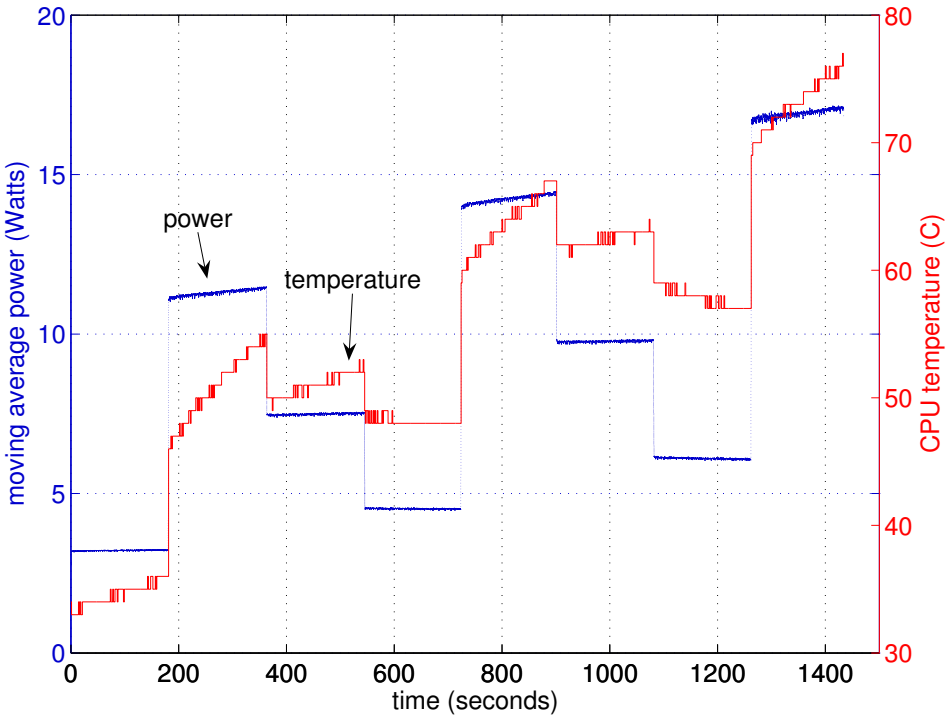


Figure 5.18: Power and temperature for `daxpy` benchmark with under-cooled conditions, with 200 seconds of each p-state in order: 600, 1600, 1200, 800, 1800, 1400, 1000, 2000 MHz.

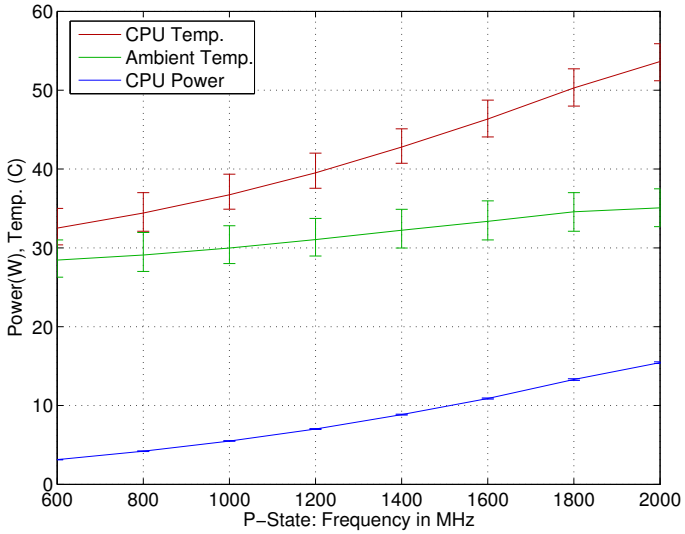


Figure 5.19: Mean power and temperature for `daxpy` repeated 10x at each p-state; minimum and maximum values charted with vertical bars. Note the insignificant variation in power yet substantial difference in temperature.

5.4 Effect of Ambient Temperature

Figures 5.19 and 5.20 show the effect of the `daxpy` benchmark repeated ten times for each p-state. The execution order was an inner loop of a single invocation of the benchmark for each p-state from the maximum frequency p-state down to the lowest-frequency p-state, and an outer loop of those eight instances repeated ten times. The benchmark repeated continuously, 80 times total, from 5:40pm through 3pm the following day. Weather conditions in the form of a cold front, verified with historical weather data [92], caused the ambient temperature of the Pentium M system to drop by about 5 degrees overnight, which over time cooled the CPU temperature by 5 degrees for a given fixed p-state and steady workload.

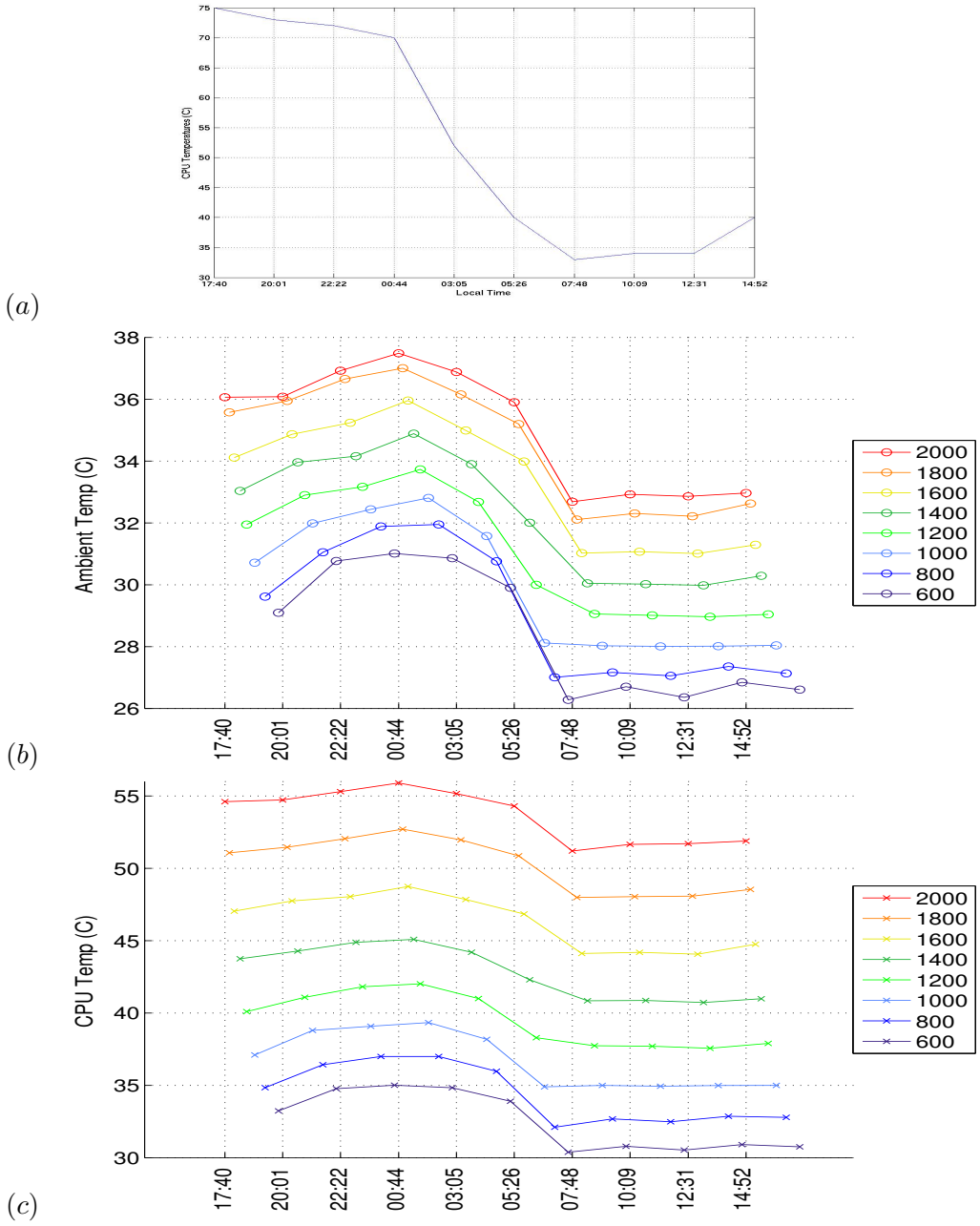


Figure 5.20: (a) Reported outdoor temperatures according to historical weather data (b) steady-state CPU temperatures (c) steady-state ambient temperatures.

5.5 Thermal Analysis with Typical Workloads

Thus far, we have examined power and thermal response to p-states with microbenchmarks that reach steady states under controlled conditions. In this section, we measure the response with typical workloads that do not necessarily converge to steady behavior. We executed the full SPEC CPU2000 (floating-point and integer) suite with a fixed p-state for the duration of the run, for each of the 8 p-states, with the fan enabled. In contrast to the steady microbenchmark temperatures, we find that CPU temperatures for the SPEC CPU2000 benchmarks fluctuate, tracking the time-varying workload behavior and power consumption. Despite the variation in CPU temperature, the ambient temperature does reach a plateau after several minutes of execution.

5.5.1 Individual Benchmarks

Figures 5.21 through 5.24 show in detail the effect of p-states on CPU temperature and power for two benchmarks with different characteristics, `mcf` and `galgel`. Each benchmark executes with every DVFS p-state, held constant for the duration of the benchmark. `Mcf` is memory bound, spending most of its time waiting for memory accesses with idle core units in a clock gated, low-power state. As a result, even at the highest frequency p-state, `mcf` consumes less than 12 Watts on average. Throughout execution, intermittent power spikes are apparent in each p-state, with greater magnitude for higher frequencies. The CPU temperatures are fairly steady for each p-state, and range from about 34 °to 46 °C. The power spikes seem to have little effect on the temperature, with thermal fluctuations of 1 °C,

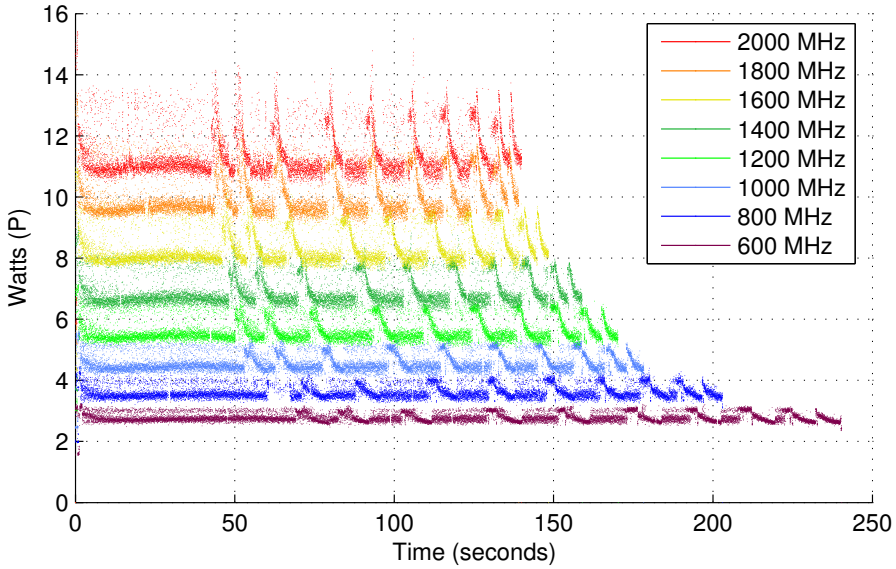


Figure 5.21: CPU power for `mcf` benchmark at each DVFS p-state.

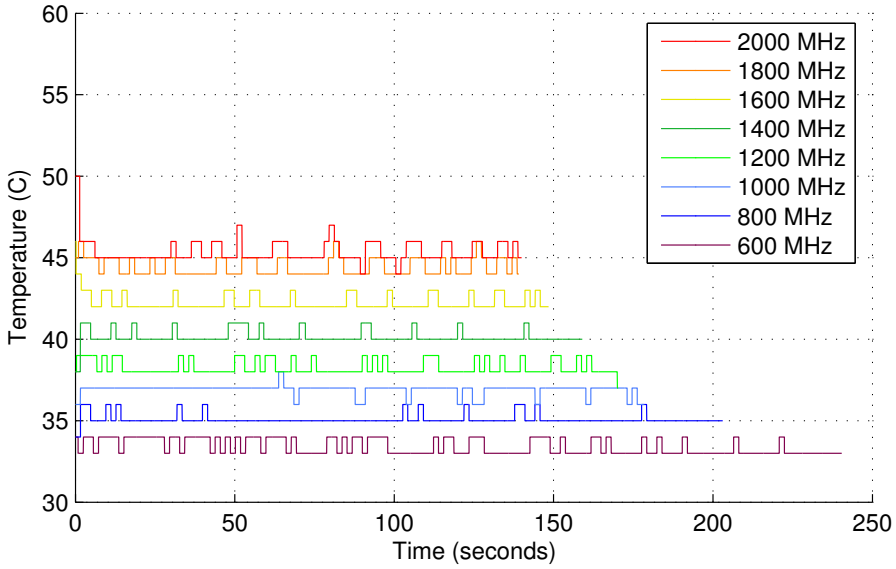


Figure 5.22: CPU temperature for `mcf` benchmark at each DVFS p-state.

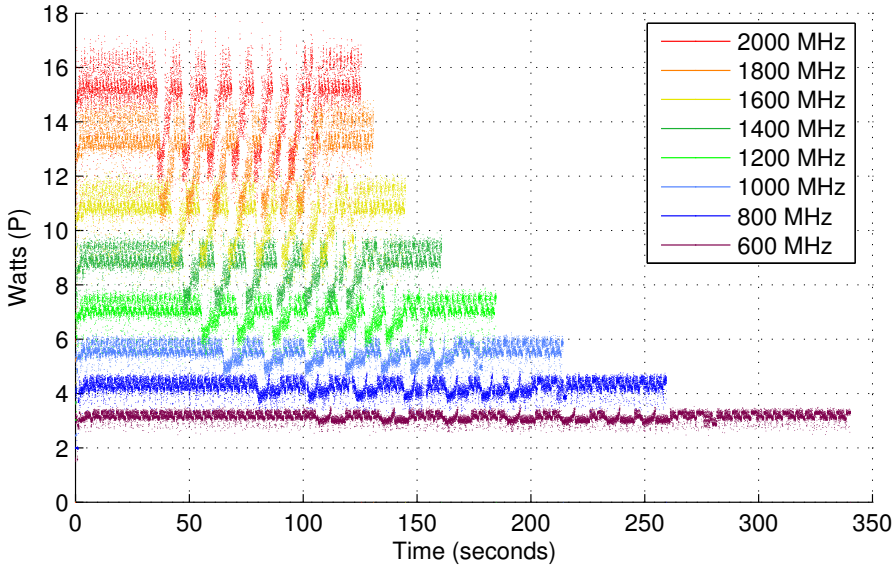


Figure 5.23: CPU power for `galgel` benchmark at each DVFS p-state.

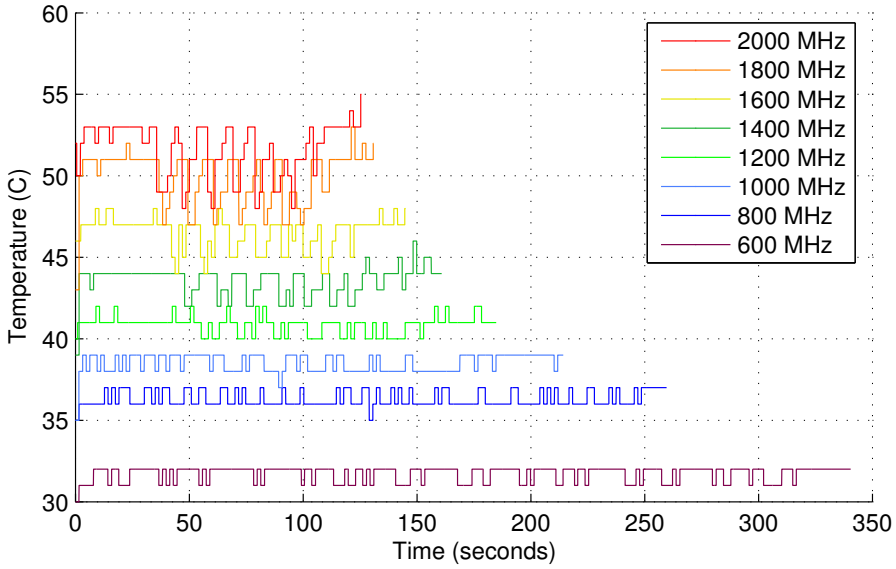


Figure 5.24: CPU temperature for `galgel` benchmark at each DVFS p-state.

imperceptible from the rounding error of the thermal sensor, for most cases.

`Galgel`, on the other hand, consumes about 16 Watts at the highest frequency due to higher levels of core activity. `Galgel` exhibits periodic high-to-low power swings during a portion of the benchmark, with a distinctive zig-zag power pattern at higher frequencies. At lower frequencies, the core processor stalls for fewer cycles at lower frequencies (the memory speeds are unchanged and lowering the core speed provides a better match between core and memory), attenuating the bursty behavior observed at higher frequencies. As a result, the zig-zag power pattern is less noticeable at the low end of the frequency range. Temperature recorded for `galgel` reflects the power trends. The temperatures for high-frequency p-states are higher than for `mcf` due to the higher heat output from higher power consumption, and the temperature fluctuations are greater at higher frequencies during the zig-zag power periods than during that application phase at lower frequencies. The temperatures range from 32 °C to 56 °C, a larger range than for `mcf`.

5.5.2 Full Suite

We execute each benchmark once, in the SPEC run execution order listed in Table 4.4 while the fan spun at a high speed, approximately 4500 rpm. We concatenate the observed data from each benchmark to reconstruct the full suite in Figure 5.25 with moving averages over 1 second for each CPU temperature, ambient temperature, and CPU power for the SPEC benchmark suite executing at a fixed p-state of 2 GHz. The ambient power ramps to a stable region within about ten minutes, while the CPU temperature fluctuates continuously within a 5-10 degree

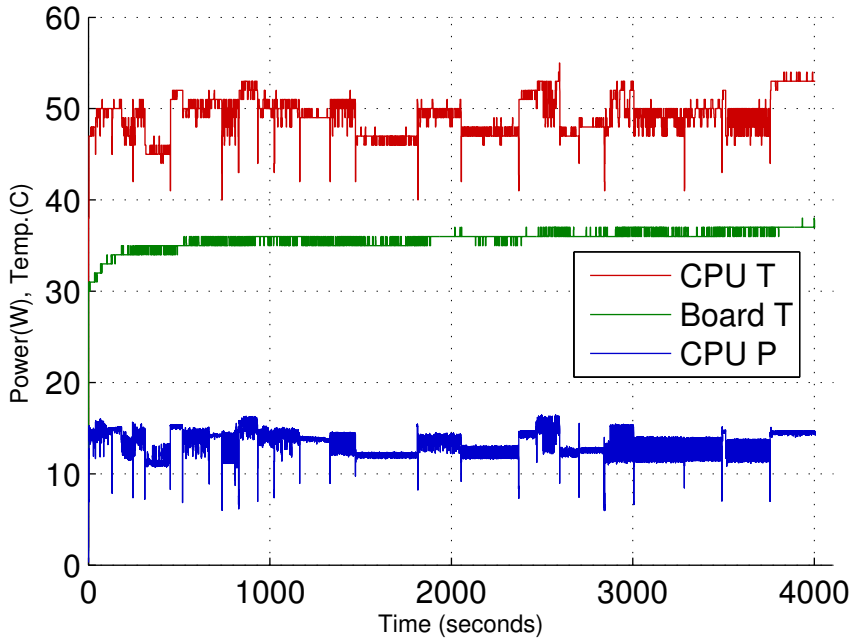


Figure 5.25: SPEC CPU2000 suite executing at 2GHz: 1-second moving average power and temperatures.

range in response to power dissipation. In the PET prototype, we take advantage of the ambient temperature's steady properties, which are less sensitive to sensor delay than the faster-moving CPU temperatures.

Figures 5.26 and 5.27 report the mean power and mean CPU temperature for each SPEC CPU2000 benchmark, at each p-state, along with the minimum and maximum recorded temperature or power values, plotted as vertical bars. In this aggregate view, we observe that temperature is less dependent upon power in the low-power range, and more dependent on power in the high-power range, similar to the observations with steady-behavior benchmarks. At 600 MHz, we observe similar power among benchmarks yet fluctuating temperature readings. At

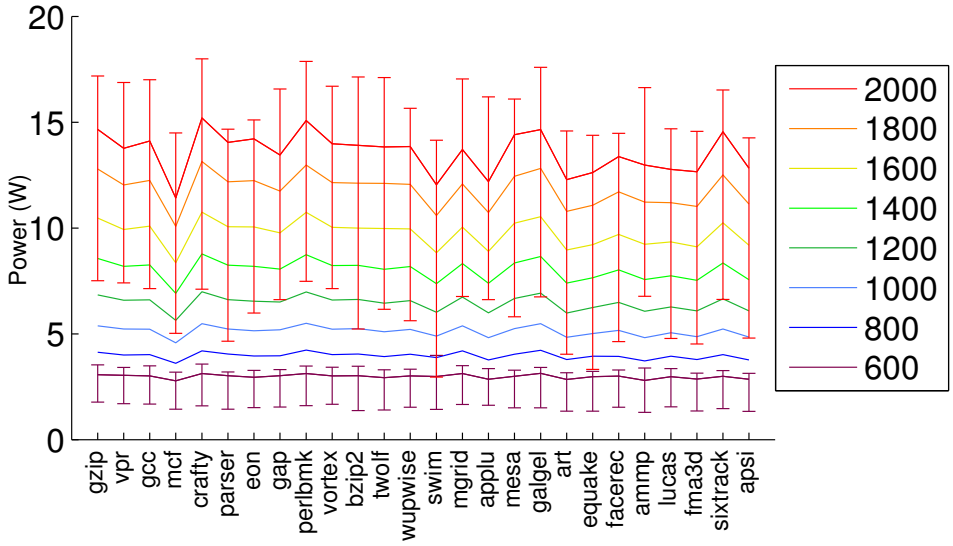


Figure 5.26: Mean power for each SPEC CPU2000 benchmark, at each DVFS p-state: 2GHz (red) down to 600 MHz (purple).

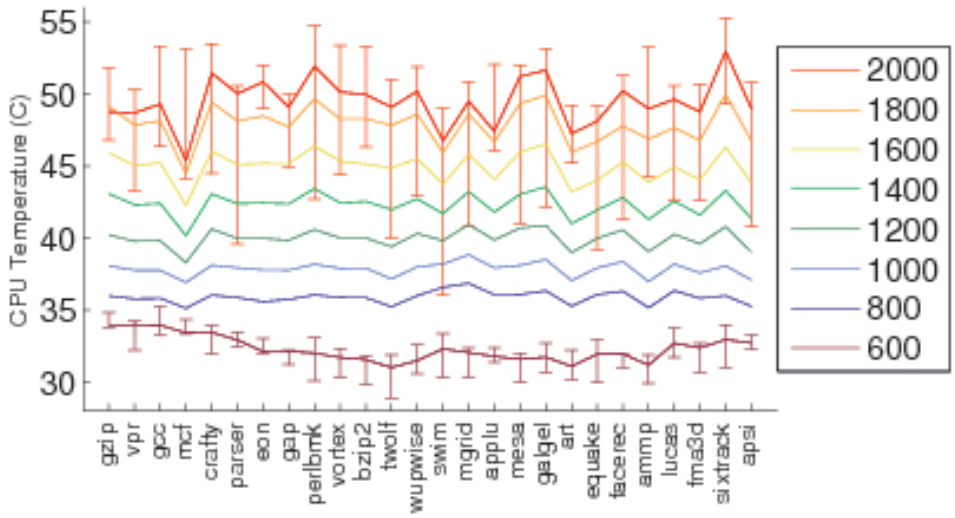


Figure 5.27: Mean CPU temperature for SPEC benchmarks at each DVFS p-state: 2GHz (red) down to 600 MHz (purple). Minimum and maximum temperatures indicated with vertical error bars.

mid-range and higher power levels, the shapes of the thermal curves more closely match the power curves. Temperature variation is larger for higher frequencies than lower frequencies, with greater minimum-maximum ranges and also larger differences between benchmark mean temperatures. It is evident that the interaction between workload characteristics and p-state influences power and CPU temperature. For example, the benchmark `mcf` at the 2000 MHz p-state exhibits a mean temperature similar to `crafty` at 1600 MHz.

5.6 Predictive Models

Predictive models provide a means to project the expected outcome of each dimension (power, performance, etc.) into the goal space, to allow the manager to choose the best operating point. In this section, we discuss predictive models for performance, power, and temperature that we developed for a PET prototype implementation.

5.6.1 Power Estimation

Several researchers have developed power and energy models based on performance events counters. Bellosa developed one of the earliest counter-based energy models, which was accurate within 10% for 2 event counters and within 5% with 5 event counters [7]. More recent extensions of this work include a technique known as *Process Cruise Control*, in which performance event counters gauge memory instructions per cycle and completed instructions per cycle (IPC) to enable the operating system to choose the optimal frequency for each process in an XScale system with

dynamic frequency and voltage scaling [89]. Yaari used multiple linear regression with synthetic benchmarks and a subset of the SPEC CPU2000 benchmark suite to develop a power model for an Intel Pentium III system, including power consumption in the memory, fan, and disk drives based on event counters [94].

Isci and Martonosi monitored power with a Pentium 4 system, developing a per-component power model for the processor core using 24 performance event metrics, obtained by rotating through a series of 4 groups of 15 counters each. They evaluate the power model by monitoring power in real time while executing several SPEC CPU2000 benchmarks, observing that the average difference per benchmark between observed and predicted power was about 3 Watts, on a system that varies between approximately 30-50 Watts [47].

Bircher *et al.* investigated the correlation of several performance monitoring counters with power for a Pentium 4 system in [8] and created an accurate model of average power with two event counters: fetched uops and microcode ROM-delivered uops. The authors note that fetched uops provide a better indicator for power than IPC (completed instruction per cycle) counters because the counter captures speculative instructions which consume power but are not included in the IPC counts. They constructed a power model with linear regression of performance event counts and power and used a subset of SPEC 2000 benchmarks to build a model, then tested the model on the full suite.

In our work, we explored several power models based on counter models and measured power, with a range of complexity and accuracy. An important distinction for our power models is the estimation at other p-states based on activity observed

at the present p-state. By predicting power at multiple p-states with sufficient accuracy, a controller could choose an appropriate setting without the overhead of exploring several settings in a test-and-set manner. Combined with closed-loop feedback control, an accurate prediction mechanism could allow a controller to prune ineffective choices to improve settling time.

Through related work and our prior power-modeling studies, we determined that performance counters provided a window into workload characteristics relevant for predicting power consumption. We developed accurate power models with limited information from two performance counters and lightweight calculations [72], [75]. However, performance prediction and power prediction require different instruction counter types to achieve suitable accuracy: *decoded* instructions to capture speculative activity for power models and *committed* instructions for performance models, in addition to a memory-access counter common to both models. Three dedicated counters would provide sufficient information for our predictive models. Unfortunately, the Pentium M processor and thus our prototype implementation is limited to two counters. Rather than toggle between counter types and lose information for one type while sampling the other counter, we set the counter event types to those required for performance prediction and we developed an alternate power model based on measured power. For these prototype experiments, we use a simple power model based on measured data with the SPEC `train` input set. The power model development process is as follows.

1. measure power with the training dataset

2. correlate power at one p-state with other p-states

3. apply linear regression to calculate model coefficients

4. evaluate model accuracy

Measure power: I measured power for each benchmark in the SPEC CPU2000 suite with the `train` input set for each of the 8 DVFS p-states. The nominal sampling rate was 100 samples per second, with the effective rate of 89.99 samples per second due to variable timing in the software timer interrupt. Figure 5.28 shows the measured power at each p-state. The fastest execution, at 2 GHz, completed in 523.6 seconds (approximately 8.7 minutes) and the slowest, at 600 MHz, at 1572.6 seconds (approximately 26.2 minutes). The `train` input set executes almost 9 times faster than the `reference` input set typically used for performance analysis yet exhibits similar workload behavior.

Correlate power: The objective in correlating power between traces is to match the same point in the program across all p-states. For the microbenchmark-based power models we previously developed [75], a single power value represented the benchmark behavior, and correlation was a simple matter of comparing mean power for each benchmark at a given p-state to the mean measured power at each other p-state. For a time-varying workload such as the SPEC CPU2000 suite, matching points between p-states is more complex. Figure 5.28 shows the non-linear relationship in execution times for each p-state; the end point of each trace (total execution time) forms a curve from the 2 GHz trace down to the 600 MHz trace. Memory

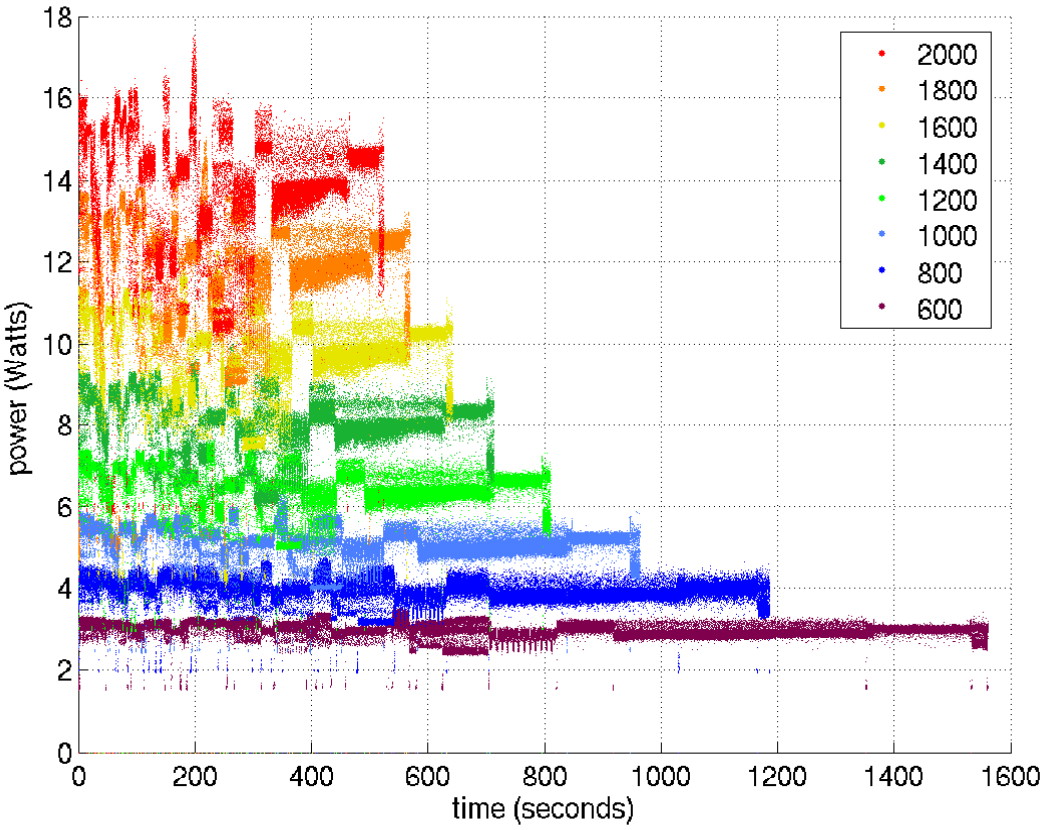


Figure 5.28: Measured power for SPEC CPU2000 `train` input, 8 p-states.

dependence causes the non-linearity. Memory accesses stall fewer CPU cycles at lower frequencies than at higher frequencies; thus, low-frequency execution requires fewer total cycles than high-frequency execution.

Resampling each trace to a common length aligns the workloads, such that any power sample in one trace may be compared to the corresponding point in the workload at another p-state. I resampled the power traces for each benchmark to the 2 GHz p-state trace length. I applied the signal processing `resample` function in MATLAB, which uses a polyphase filter implementation to selectively remove and interpolate data points throughout the trace, compressing the trace in time while preserving the magnitude and phase characteristics. The extent of compression depends upon the original trace length. I chose a resampling method that preserves the variations in memory dependence by resampling each benchmark individually, rather than resampling the full suite together. The process does not preserve the intra-benchmark memory dependence variation, but we expect the intra-benchmark variation to be negligible for this suite. Figure 5.29 shows the resampled traces, each approximately 523 seconds in length.

Linear regression: With the p-state power traces aligned in time, we can compare power consumption by p-state throughout the suite. I used linear regression to find the relationship between power at each p-state in the form of:

$$\hat{P}_y = \alpha_{xy} P_x + \beta_{xy}, \tag{5.1}$$

where P_x is measured power at the p-state x, α_{xy} and β_{xy} are coefficients

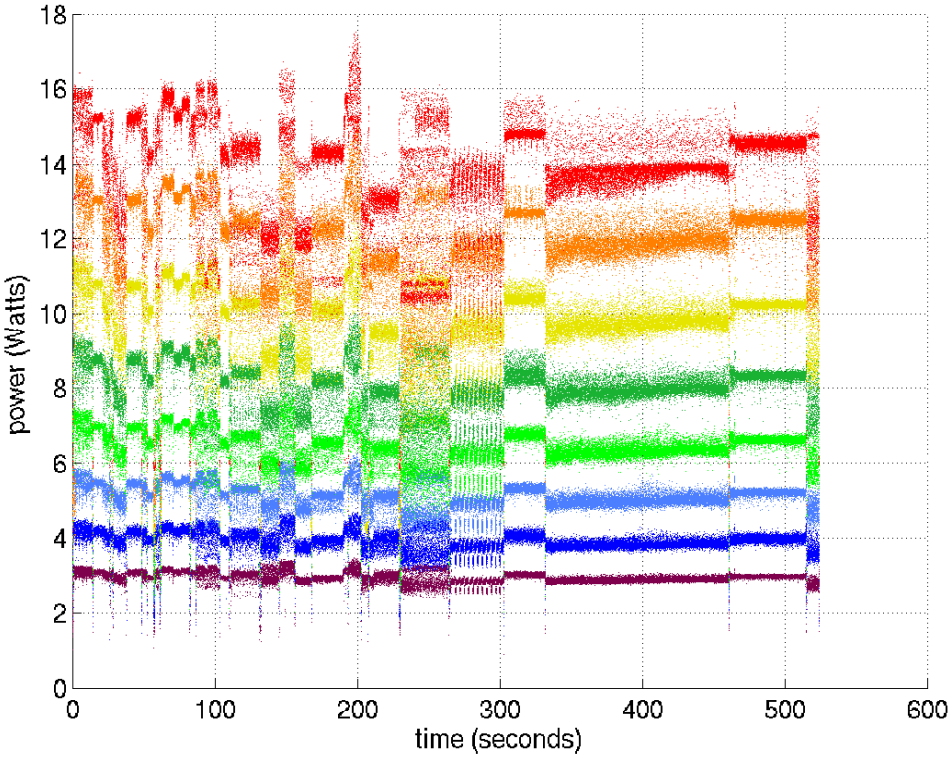


Figure 5.29: Resampled power traces for SPEC CPU2000 `train` input, 8 p-states. Resampling aligns workload behavior is aligned in time while preserving power characteristics.

to estimate power from p-state x to p-state y, and $\hat{P}$ is the estimated power for p-state y. The regression process produced coefficients for α and β for each of the 64 combinations of DVFS and clock throttling p-states, a separate set of coefficients for estimating power from each p-state, to each p-state. The set of coefficients includes predicting power at the same p-state as the measured data, which indicates the inherent spread in power for the workload at that p-state.

Evaluation We first examine the limits of model accuracy by comparing the linear model with the dataset used to create the model. We compare each recorded power sample with the power estimate for that sample. It is important to compare each sample point in order to detect both overshoots and undershoots that would average together to form a misleading smaller total error if the data were compared on a per-benchmark basis or other long period of time.

Figure 5.30 displays the linear regression residuals (model errors) for each recorded sample in a cumulative distribution function format (CDF), with one CDF line for each of the 64 combinations of 8 p-states. Most samples' residuals are within +/- 2 Watts. The model fit demonstrates that less than 10% of all points are overestimated (negative errors) by 2 or more Watts, and less than 4% of all points are underestimated (positive errors) by 2 or more Watts. Overall, overestimates occur less often, approximately 30% of the error distribution, but tend to be larger in magnitude than the underestimates.

Overestimates can cause performance loss when a power manager chooses lower frequency p-states to meet power budgets when a higher frequency may have

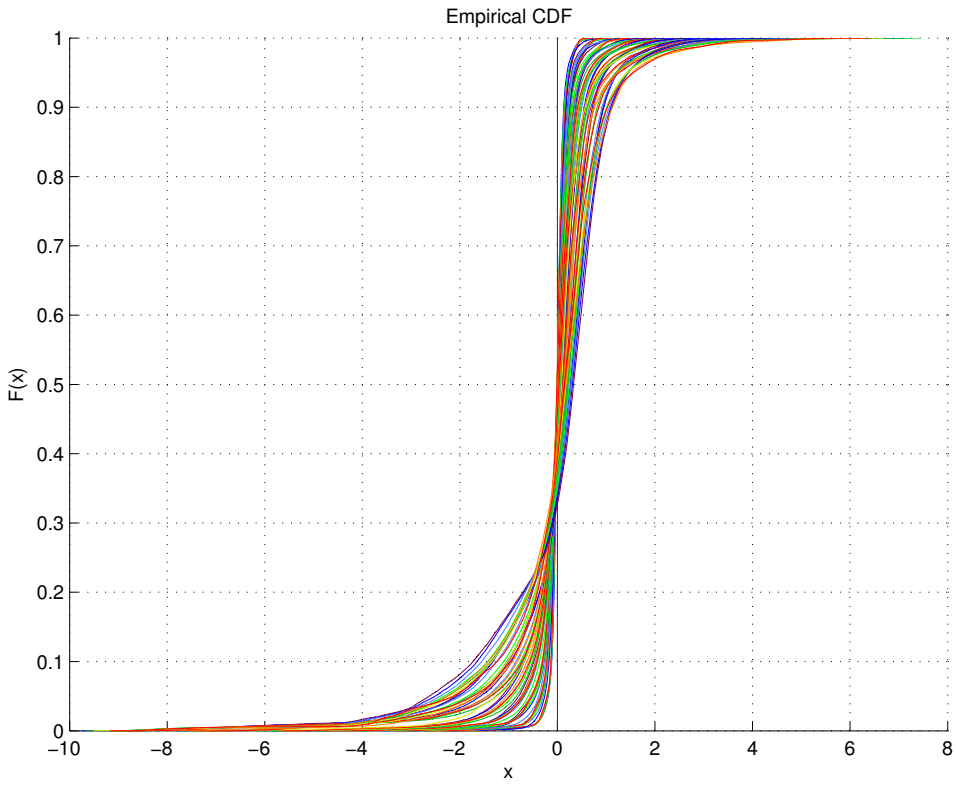


Figure 5.30: Cumulative distribution function (CDF) for power model regression residuals. The x-axis is the model error in terms of power (Watts); positive x values indicate underestimates and negative x values indicate overestimates.

been suitable. Underestimates can be detrimental to maintaining power within a power budget, when a manager uses a p-state that is expected to be within the allowed power envelope yet actually violates the power budget. Based on the model fit, we expect that a manager will be able to project power for all p-states with reasonable accuracy in the vast majority of points for workloads that have similar power characteristics as the training data set, SPEC CPU2000 `train`.

Figures 5.31 and 5.32 show subsets of the residual fit data to illustrate the

frequency dependence of model accuracy. In this system, power for lower-frequency p-states is more easily predicted for two reasons. First, memory-dependent workload variation is attenuated by fewer stall cycles for memory accesses due to the lowered core frequency. Workloads that are mildly memory-bound at higher frequencies behave more like core-bound workloads at lower frequencies. Second, at low-frequency p-states, the frequency-independent power of about 2 Watts dominates the total processor power. The combination of these two factors yields very similar power behavior for low-frequency operation, even for workloads that exhibit a large degree of variation at higher frequencies.

Due to the small range of power values, estimating power on the training data set and at the same p-state, from 600 MHz to 600 MHz, yields nearly perfect prediction. Figure 5.31 shows the regression residuals for the model's best case, estimating the 600 MHz p-state power from measured data at any p-state. A beneficial aspect of good model accuracy at low frequencies is the ability to meet strict low power budgets.

Predicting the larger range of power for higher frequencies based on low-frequency measured data creates a larger model error. Figure 5.32 shows model residuals for estimating the maximum 2000 MHz p-state power from data sampled at other p-states.

Unlike the low-frequency case where the model fit is nearly perfect for the 600-to-600 MHz prediction (indicated by the vertical line at 0 Watts of error in Figure 5.31), predicting 2000 MHz operation from measured data recorded at 2000 MHz on the same training set does not demonstrate a close model fit. The model fit for

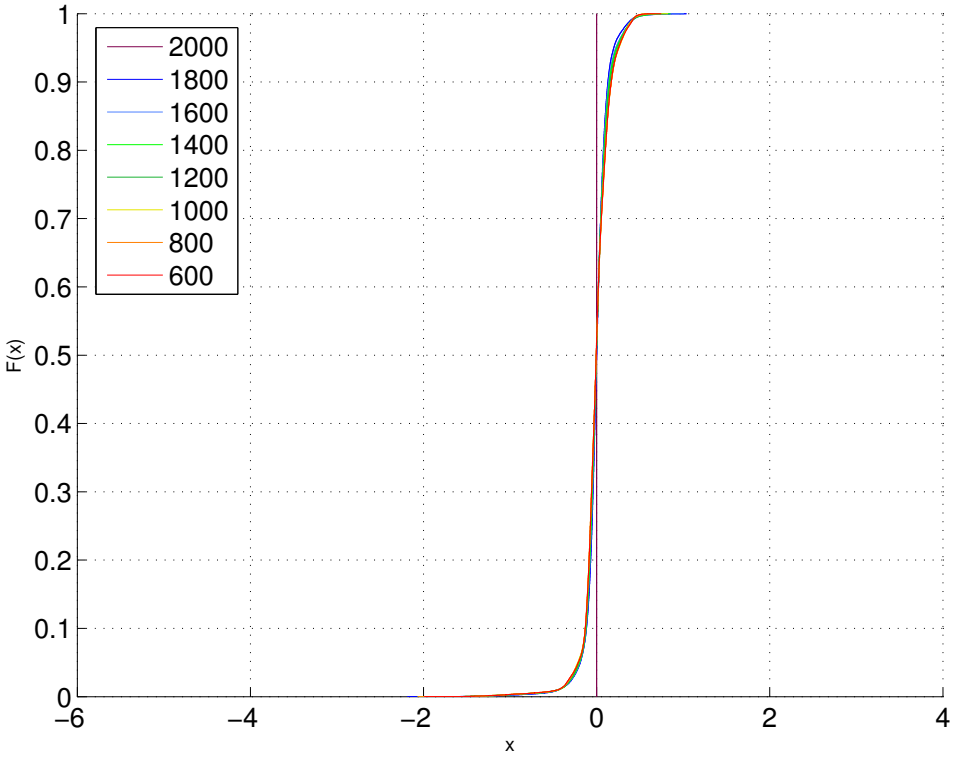


Figure 5.31: CDF for regression residuals for estimating power for the 600 MHz p-state, from all p-states. The x-axis is model error in terms of power (Watts). Model errors for the 600 MHz p-state are small, indicating a good fit between the data set and linear equation model.

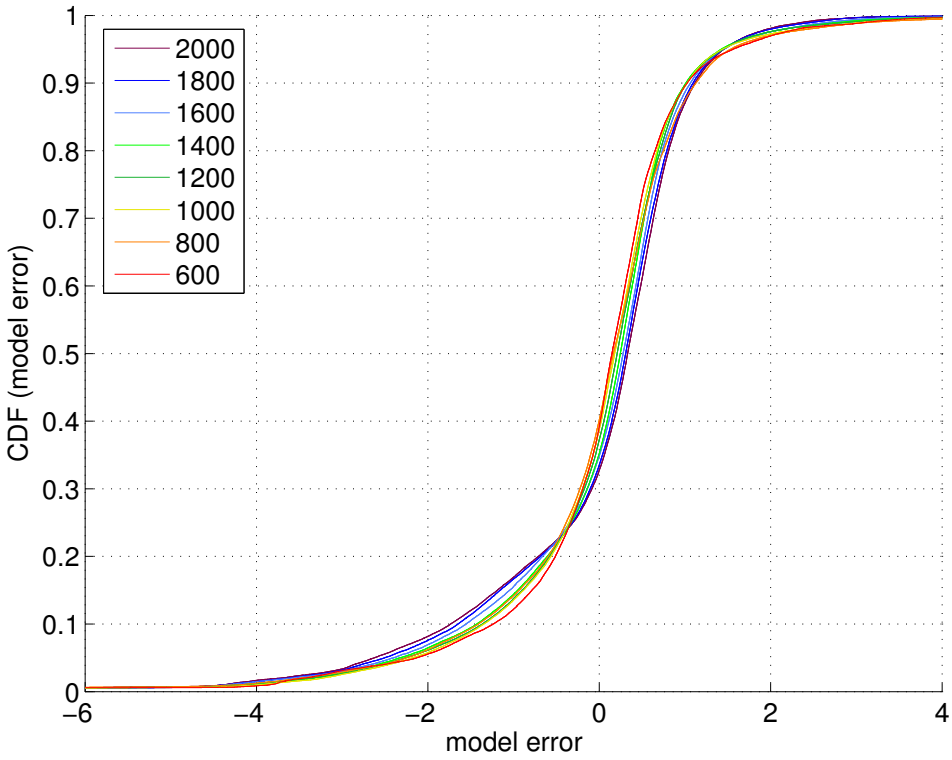


Figure 5.32: CDF function for regression residuals for estimating power for the 2000 MHz p-state from all p-states. The x-axis is model error in terms of power Watts. The larger model errors indicate that the linear model does not describe the data set at 2 GHz as closely as the 600 MHz p-state in Figure 5.31.

estimating the 2000 MHz p-state power from other p-states shows a wider spread of error for all p-states.

The dataset's inherent spread in power for the highest-frequency p-state limits accuracy. This power model is acceptable for the PET manager prototype because our system can tolerate short excursions beyond power or performance constraints in the rare events that power mis-prediction leads to a poor p-state choice. In situations that require even higher degrees of model accuracy, a hybrid model of event counters and measured power may provide information to more fully describe the relationship between p-state, workload, and power consumption.

5.6.2 Temperature Estimation

We applied our observations of thermal response to DVFS to develop a thermal estimation model that predicts the CPU temperature response to changing p-states based on current conditions, for use in a power-temperature controller. We applied linear regression to the empirical steady-state ambient and CPU temperatures and power for microbenchmarks measured at each p-state to create a thermal model. The model captures the effects of both environmental conditions and power consumption on the CPU temperature:

$$T_{CPUest} = \tau P + T_{ambient} \tag{5.2}$$

where T_{CPUest} is the estimated CPU temperature, τ is a scalar coefficient, P is the processor power at a given p-state, and $T_{ambient}$ is the current ambient temperature. Linear regressions indicate that the coefficient τ varies slightly by

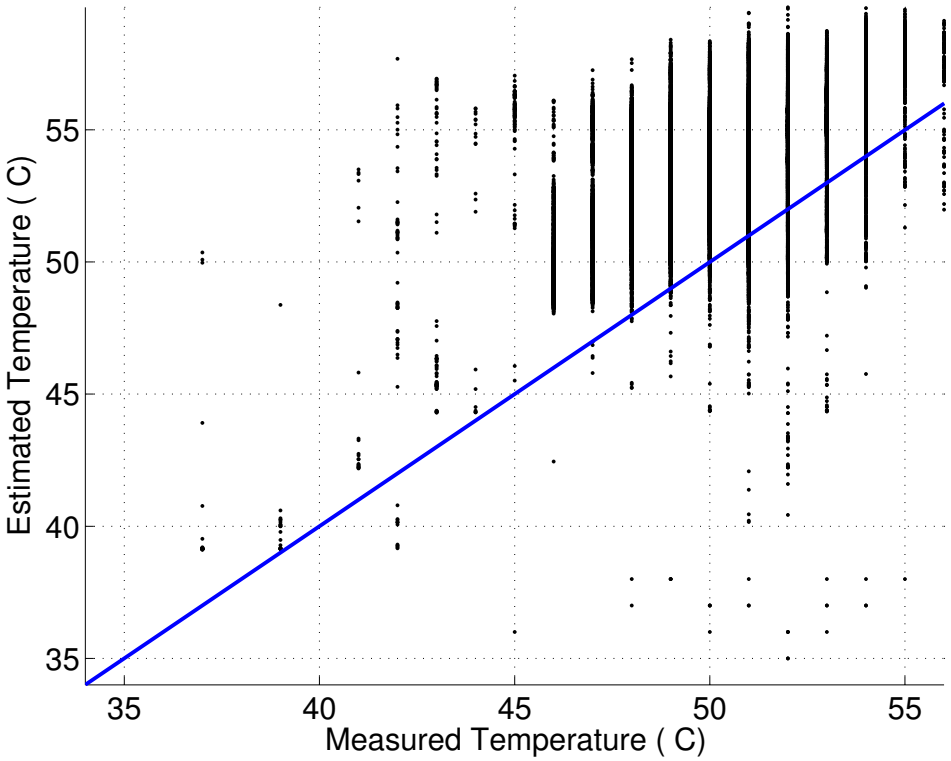


Figure 5.33: Comparison of estimated and measured CPU temperature for SPEC CPU2000 benchmarks at the 2 GHz p-state.

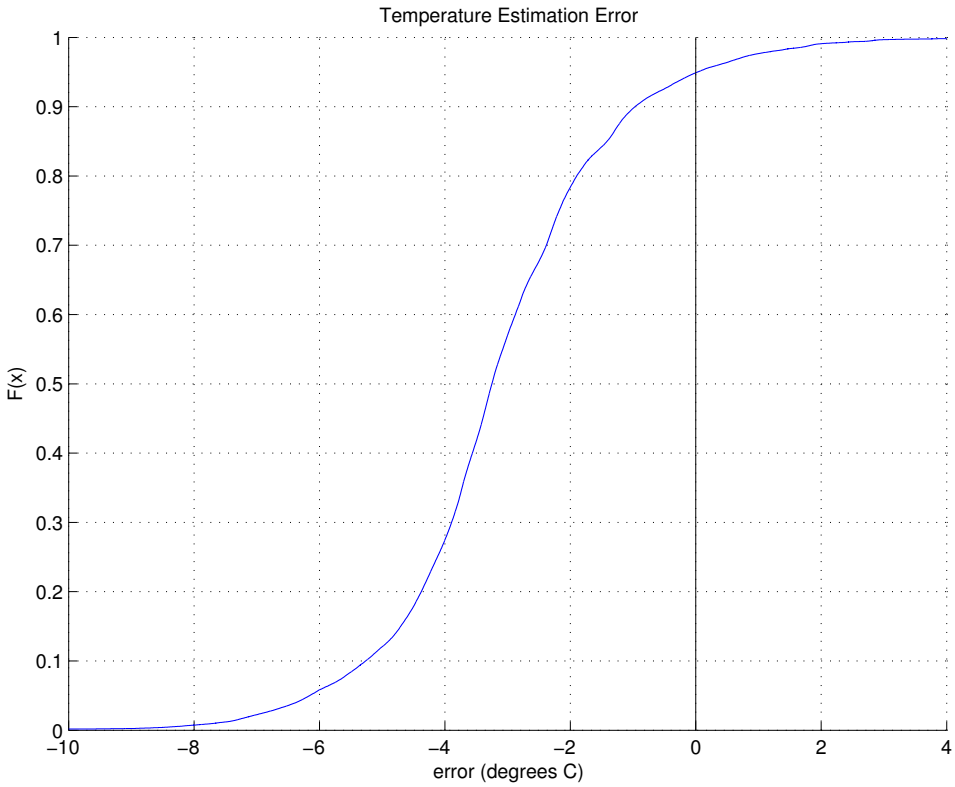


Figure 5.34: Cumulative distribution function (CDF) of temperature estimation error for SPEC CPU2000 benchmarks at the 2 GHz p-state.

benchmark; we surmise that the difference is due to the single CPU thermal sensor that is spatially closer to the hotspots of some workloads than others. In our work, we simplify the equation to use a fixed constant of $\tau = 1.25$.

Other forms of a predictive thermal model would also be possible, such as directly predicting CPU temperature for other p-states given the current CPU temperature. The form of Equation 1 proved useful by leveraging our prior work that estimates power at all p-states based on measurements for the current p-state [72]. By using predicted power in Equation 1, we are able to quickly project CPU temperature for all p-states. The thermal model also exploits the slow rate of ambient temperature change. In systems with infrequent measurements or a long delay for temperature sensor readings, a slow-moving reference point in the estimation model such as the ambient temperature better tolerates sensor delay than a quickly changing measurement such as the CPU temperature.

Figure 5.33 charts predicted versus measured T_{CPU} for each recorded datapoint in the execution of the SPEC CPU2000 suite at 2 GHz. The diagonal line represents a perfect prediction; above the line is an over-estimate and below the line is an under-estimate. The figure shows that over-estimates occur throughout the temperature range, while under-estimates are more concentrated toward higher measured temperatures. Figure 5.34 charts the error data with a cumulative distribution function to show the distribution of over-and under-estimates overall. The thermal model under-estimates in less than 5% of samples, with an average of 1.3 °C for underestimates. The model over-estimates T_{CPU} in 95% of all samples, with a mean of 3.4 °C for overestimates. The bias toward overestimates stems from the

model training dataset of high-activity benchmarks that produce higher T_{CPU} values than the SPEC CPU2000 workloads, and is useful for situations that warrant a conservative estimate. More aggressive models could shift the error toward a more balanced over- and under-estimation and rely on the built-in thermal safety features in the event of a grave mis-prediction.

5.6.3 Performance Estimation

The prototype PET manager uses a performance metric of instructions per second, IPS, gleaned from hardware event counters (also known as event counters or performance monitoring counters) The prototype software applies a non-linear performance estimator developed by Rajamani *et al.* to predict performance at all p-states based on measured IPS at the present p-state [72]. The prediction first calculates estimated instructions per cycle, IPC, and then multiplies by each frequency to determine IPS. The equation to estimate performance at p-state y from measured performance at p-state x is composed of two components, a core-bound case and a memory-bound case. For core-bound applications, IPC is constant, regardless of core frequency. Memory-bound behavior is more complex, and the model uses a nonlinear equation that captures the effect of the difference in frequencies to scale performance to other p-states.

Core-bound, for $DCU/IPC < 1.21$

$$\hat{IPC}_y = IPC_x \tag{5.3}$$

Memory-bound, for $DCU/IPC >= 1.21$

$$\hat{IPC_y} = (f_x/f_y)^{0.81} \tag{5.4}$$

In the prediction equations, $\hat{IPCy}$ is the predicted performance at frequency y. f_y and f_x are the frequencies at p-state y and x, respectively. The exponent 0.81 was determined empirically with the MS-Loops microbenchmarks.

The performance counter DCU (`DCU_MISS_OUTSTANDING`) tallies the number of cycles in which the data cache unit (DL1) waits for cache misses to be serviced by higher levels of the memory hierarchy. My contribution to the performance estimator was applying the counter ratio DCU/IPC, which approximates a proportion of work performed to the amount of time stalled, to serve as an indicator of memory boundedness for workloads executing on the Pentium M system. I originally used the ratio for a counter-based power model, and we later used the counter ratio in the performance model, as well.

5.7 Summary

Current and next-generation commercial computing systems have access to several power management techniques, and to employ them effectively, we need to understand the effects on performance, power and temperature. We characterized application response to two power management techniques, DVFS and clock throttling, applied individually and in combination on a Pentium M system with a set of microbenchmarks designed to capture core and memory subsystem activity. We analyzed microbenchmark power and performance for three dataset sizes corresponding

to core-bound, intermediate, and memory-bound programs. We found that for techniques applied individually, DVFS is more effective than clock throttling at reducing power while preserving performance.

We also applied both techniques together in each of the 64 combinations of 8 frequencies and 8 clock throttling levels. We demonstrated that a performance or power target can be achieved with multiple combinations of DVFS and clock throttling settings; however, DVFS provides better power-performance efficiency than clock-throttling.

We investigated the thermal response to DVFS of an Intel Pentium M system. The system's thermal response is determined by two factors in opposition: heat generated by power dissipation, and heat conducted by the cooling system. In this section, we observe the effects of power on temperature under maximum cooling conditions and the effects of the cooling environment on CPU temperature under steady-power conditions, with the steady-behavior microbenchmarks of the `MS-Loops` suite.

- We demonstrate that CPU temperatures scale with DVFS p-states under well-cooled conditions, for a given workload activity and ambient temperature.
- We identify the two-stage thermal response to p-state changes: a quick thermal change (milliseconds) followed by additional drift after the local air temperature adjusts to the new CPU temperature (minutes).
- We demonstrate a linear relationship between power and temperature, in a well-cooled environment.

- We develop a simple thermal estimation model based on current observed conditions to predict the effect of DVFS options on CPU temperature.

Our results show that DVFS has a strong and immediate influence on processor temperature and confirms that DVFS could be a viable thermal control mechanism. However, CPU temperature is also affected by workload activity and cooling environment, thus highlighting the need for accurate and timely thermal sensor data to reflect current conditions for use in dynamic thermal management.

With SPEC CPU2000 benchmarks executing at a fixed 2 GHz frequency, we observe that power and CPU temperature fluctuate with workload activity, while the ambient temperature reaches a plateau. We measured temperature and power for each benchmark at each DVFS state, and observed that the mean and range of power and CPU temperature depends upon the workload. We also note that power alone does not predict CPU temperature, due to the ambient influence.

We developed estimation models for performance, power, and temperature for the Pentium M platform to use with a PET p-state selection algorithm. The models predict the outcome for each p-state based on the current p-state, allowing unacceptable choices to be pruned from the selection algorithm.

Chapter 6

Dynamically Managing Performance, Power, Energy, and Temperature

6.1 Overview

Computing systems are typically equipped with dynamic voltage and frequency scaling, clock throttling, power-down modes, and other techniques that control power and temperature at the expense of possible performance degradation. Current solutions to managing power for high performance computing systems are often built for power and thermal envelopes, with the goal to maximize performance within an upper bound on power/thermal consumption. The power limit is designed to ensure that power-related effects—overall heat dissipation, localized hotspots, maximum current draw, total energy consumption, etc.—are within an acceptable range. For example, Foxton dynamically changes frequency and voltage to tune temperature and power consumption to the desired level [62].

6.1.1 Problem

Simply maximizing performance within a power-thermal limit, however, does not ensure either high performance or effective power management. For example, a memory-bound application running at the maximum allowed power level may show only modest performance improvements compared to executing with lower frequency and considerably lower power. A more comprehensive approach to power

and performance management could direct such an application to execute at a lower power-performance point, allowing the hardware to cool and reserving energy for a burst of activity that would significantly benefit performance.

One alternative is to consider performance, power, and temperature when selecting power management settings. Researchers have studied power-performance tradeoffs extensively and have proposed combined power-performance metrics such as the energy-delay product (EDP), millions of instructions per Watt (MIPS/Watt), energy per instruction (EPI) and several application-specific variations that weight performance or power terms.

The primary limitation of these approaches is that combining both terms into a single metric obscures relevant information. A high-performance, high-energy scenario can be equivalent to a low-performance, low-energy scenario in EDP values. Even with weighted terms, settings with different characteristics may be equivalent in the combined metric.

6.1.2 Solution

We invented a new approach to solve the multi-dimensional, multi-goal problem. We named our framework **PET**, an acronym for Performance-Power-Energy-Temperature, which were the primary constraints in our work. The approach is also suitable for managing additional constraints such as current draw, soft errors, etc. The key principles of the PET approach are:

- multi-criteria objectives

- goal-driven decisions
- coordinated mechanisms
- continuous, run-time optimization.

This chapter explains each principle and the benefit it provides for coherent management.

6.2 Multi-criteria Objectives

Power supply is limited in large systems; however, it is not the only concern. Heat density, junction temperature, total energy consumption, and other related effects each impose unique requirements and constraints on system design. Managing one aspect does not guarantee a desirable response in others. For example, reducing power by lowering the frequency and voltage may degrade performance and can cause the total energy to increase due to longer execution time; minimizing energy by completing tasks quickly improves performance at the expense of exceeding power limits. Avoiding thermal emergencies by throttling the clock can drastically reduce performance without a significant reduction in power.

It is a rare case that the desired outcome fits neatly into a single objective. Even a straight-forward mandate of achieving maximum performance is constrained by physical limitations of power supplies and fans or liquid-cooling capabilities. Within a system design, priorities among competing objectives vary by installation, depending upon cooling capacity, power supply, quality of service agreements, budget, and other factors. Priorities also shift over time: gradually as server rack

population increases, intermittently as customer demand fluctuates, or abruptly due to fan or power supply failures. The manager must translate nebulous desires such as "low operating cost" and "high performance" to monitored and controlled aspects of the physical system, such as voltage, frequency, and instructions completed per second.

The result is a complex web of inter-related constraints and requirements. We develop the PET framework with a multi-dimensional approach to solving this inherently multi-dimensional problem, by providing a means to convey multiple expectations, such as operating cost, quality-of-service requirements, thermal capacity and power supply constraints to controllers within the system that dynamically tune operating states to best meet those demands.

6.3 Goal-driven Decisions

The goal-seeking approach is flexible, unlike decisions that react to specific events with pre-determined responses, such as the Pentium 4 thermal control policy paraphrased as "if temperature exceeds the threshold, then throttle the clock." A goal-driven management approach is able to adapt to a wider range of operating conditions and resource use, allowing the processor to run closer to the edge of power, temperature, and energy limits. For example, the manager could choose to enable clock gating in a thermal emergency or migrate an active thread to another core, or simply reduce the frequency and voltage levels. The goal-seeking approach can also ensure safe operating conditions for run-time environments and configurations that are not anticipated during design and validation phases.

Instead of issuing specific directives to local managers, such as "decrease frequency and voltage during periods of low CPU utilization," we employ a goal-driven approach where the local manager is aware of the desired outcome, and it determines suitable p-states states based on local information. For a processor-level manager, for example, the information would include workload behavior and CPU temperature.

6.3.1 Goal Specification

One obstacle to effective management is translating the high-level desired outcome to measured and controlled quantities available at a lower level for local decisions. Our solution is a concise goal syntax that encapsulates a small set of specifications in terms of constraints and requirements. The manager interprets the goal in the context of present workload and environmental conditions.

One option would be conditional *ceretis paribus* preference nets (CP-nets), which represent preferences such as "I prefer red wine to white if meat is served," meaning that if everything else is equal and meat is served, red wine is preferable to white wine [14]. This type of preference expression would map well to management macro objectives, such as "I prefer to save operating costs rather than execute as fast as possible, if performance targets are met." With this formulation, it is possible to prioritize performance over power, and yet also save power when possible. However, determining optimal solutions for CP-nets is NP-hard.

Domshlack *et al.* describe a means of approximating CP-nets to combine hard and soft constraints with qualitative user preferences, in linear time. They

build a constraint graph and compute preferences and weights for each constraint. We follow a similar course, also combining hard and soft constraints with user preferences, although we map qualitative preferences, such as "lower operating cost" or "higher throughput," to quantitative preferences that reflect the capabilities and physical bounds of the system in the goal. A potential disadvantage to specifying the goal with quantitative preferences is over-constraining the outcome during the translation between qualitative and quantitative objectives, as the mapping process translates loosely specified qualitative preferences into Watts, Joules, degrees Celsius, and throughput metrics. To alleviate the potential for over-constraint, the mapping may be programmable, tailored to each hardware-software layer and updated as environmental conditions change.

For PET managers, we decompose the optimization problem into 3 steps:

- find the set of solutions that satisfy hard constraints
- find the set of solutions that satisfy soft constraints
- apply an objective function F to choose one solution.

Table 6.1 lists the goal parameters for hard and soft constraints, which take the form of minimum and maximum values for each dimension. Both minimum and maximum values are specified for completeness; in typical application, either minimum or maximum would suffice with an implicit assumption of zero power or infinite performance as the minimum or maximum, respectively.

Hard constraints form boundaries to maintain operation within acceptable

	Hard Limit		Soft Limit	
	Minimum	Maximum	Minimum	Maximum
Performance	α %	β %	γ%	δ %
Power	ε Watts	ζ Watts	η Watts	ϑ Watts
Energy	ι Joules	κ Joules	λ Joules	μ Joules
Temperature	ν C	ξ C	π C	ρ C

Table 6.1: Example goal parameters for hard and soft limits for performance, power, energy, temperature.

limits. Hard constraints are not the physical limitations of the machine; the distinction is that hard constraints delineate an acceptable region within the physical limits. For example, a chip may be capable of consuming 25 Watts of power, yet because of power supply limitations or energy budgets in a given system, the *acceptable* limit could be only 14 Watts. Hard limits on performance specify the **minimum acceptable performance**, also known as a **performance floor**. For example, quality of service agreements may dictate minimum throughput and although the system is capable of executing at a slower speed, the hard limits specify the minimum throughput needed to satisfy the performance target. Hard limits in the goal specification may change through time as performance expectations change, power and cooling systems are upgraded or experience failures. It is possible to exceed hard limits if the manager selects an unsatisfactory operating state. It is important to minimize duration and extent of excursions beyond hard limits to protect the system and meet performance goals.

While hard limits outline the boundaries of *acceptable* behavior, soft limits outline the boundaries of *preferred* behavior. The manager attempts to satisfy the soft limits, although excursions beyond the soft constraints are acceptable (within

hard limits). The variation inherent in real systems poses a challenge to always meet limits in all situations, and relaxing the enforcement of soft constraints provides a means to trim the margin, allowing less conservative operation around the preferred region, yet still enforcing the stricter hard limits.

Continuing the previous example, a soft limit of 12 Watts within the hard limit of 14 Watts allows the preference of lower power consumption for lower-priority tasks, or to save power following a burst of speed, etc. If the manager chooses an operating point that leads to power consumption of 12.75 Watts, there is no harm to the system or performance contracts; the manager simply corrects the overshoot at the next monitoring interval and continues.

The objective function F that selects one solution from the set of options that meet both hard and soft constraints may be a function of any or all of the dimensions. In many applications, a simple one-variable function, such as "maximize performance" or "minimize energy" would suffice. More sophisticated algorithms could evaluate formulas of multiple variables to optimize energy efficiency, performance per Watt, or arbitrarily complex equations.

Special Cases: It is possible that multiple operating points will be equivalent after applying the 3 steps of hard constraints, soft constraints and the objective function, in which case a manager could rely on cost functions, such as stall time required for a setting change, to choose the best option. In the event that no settings meet the goal, the manager retreats to a "safe mode" and signals an error. Safe-mode settings can either be pre-set (such as choosing a minimum frequency) or

be determined dynamically from inferring priorities from the objective functions. Implementations details differ; the common point is that if the current manager does not have the capability to resolve the problem, it takes immediate action to the safe mode while indicating a problem with an error signal.

Decoupled Objective Function: It would be possible to combine the hard and/or soft limits and objective function into a combined expression. For simple objectives, a combined expression provides a compact means to relate multiple criteria. For example, Kephart *et al.* convey their preferred target for performance and power with a single utility function that maximizes performance gain minus a power cost [51].

Decoupling the objective function from the limits allows simple expressions for complex objectives, and can reduce the manager's computational overhead. For example, an objective function that minimizes power may be used in conjunction with soft limits that prefer high performance; the outcome will be a minimal-power solution that meets high standards for performance targets. A single utility function could achieve the same result with weights or step-wise functions but the expression would be more complex and could require more calculation to determine the desired operating point as every alternative is evaluated with a complex function. In our work, we chose to decouple the objective function from the limits. The manager is able to apply the hard and soft limits to first prune illegal and undesirable operating points from consideration, then calculate the benefit of the remaining points according to an objective function.

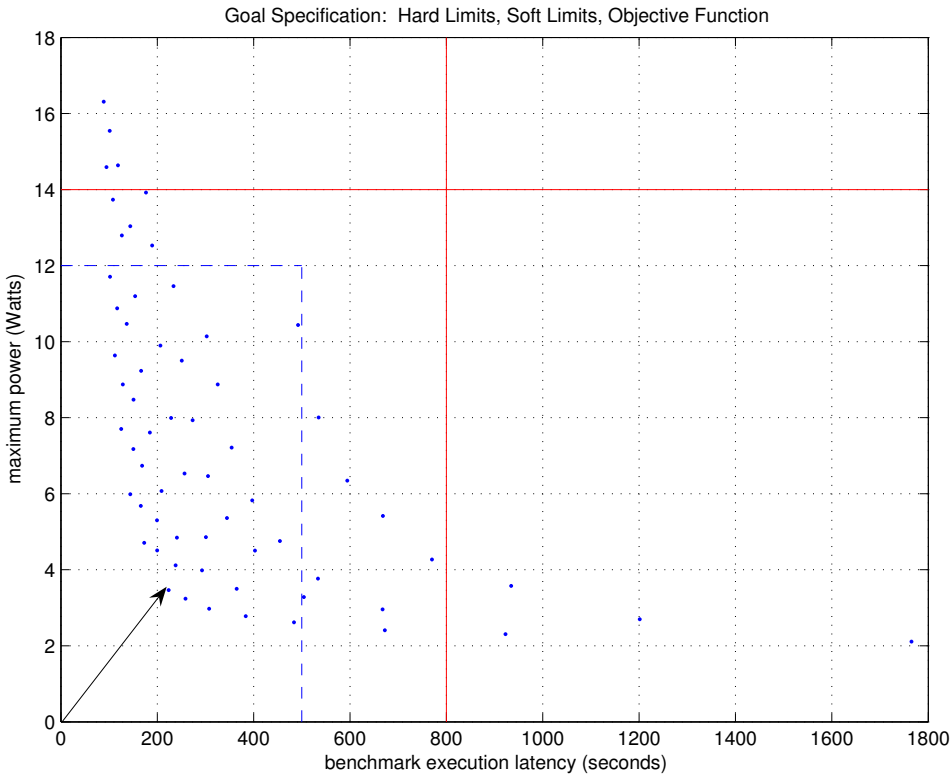


Figure 6.1: Goal specifications: solid hard limits, dashed soft limits, and arrow objective function (minimum Euclidean distance in power-performance space).

6.3.2 Goal Example

The specifications of hard and soft limits are independent for each dimension. Together, the constraints create slices and volumes in multi-dimensional space. Figure 6.1 illustrates how the goal specifications can be graphically mapped in a two-dimensional power-performance space. In this case, the performance metric is latency for the full program execution; lower latency is better. Power and performance outcomes for each of 64 possible operating states are plotted as points

in the space. Generally, higher performance requires higher power and low power consumption necessitates lower performance, although some points exhibit better power-performance trade-offs than others. The manager's job will be to select the most appropriate of the available points according to the goal target.

In the figure, solid red lines mark hard limits for power and performance (minimum power and maximum performance limits of zero and infinity are not shown). Blue dashed lines form the boundary for soft constraints. An arrow illustrates an objective function of minimizing both power and latency by selecting the minimum Euclidean distance to the origin of ideal zero power and zero latency The arrow length is equal to $\sqrt[2]{x^2 + y^2}$, where x is latency (x-axis) and y is power (y-axis).

6.3.3 Benefits

The goal specification provides two main benefits. First, it provides a common interface that allows decisions to be delegated to appropriate hierarchy levels. For example, rather than directly mapping a task state to a particular frequency in the operating system, the operating system conveys its expectations via goal specifications to a processor core manager, which chooses the best frequency. At each level in the hardware-software stack, a manager agent provides the goal specification to lower levels. In return, the lower-level managers respond with their locations in the multi-dimensional goal space. In a large-scale system, goal specifications would be determined by system administrators or automatically updated by hypervisors, virtualization applications, or power-managing utilities such as PowerExecutive [41].

Second, the goal specification provides flexibility that contemporary systems lack. Foxton Technology's "maximum frequency within power and thermal limits" policy adjusts to programmable power and thermal limits; however, it will always choose the maximum frequency that meets the limits, regardless of the performance benefit. For memory-bound workloads, higher frequencies consume more power without significant performance improvement. Our goal specification allows the manager to distinguish between Foxton's "maximum performance at any cost" and cost-effective alternatives like "best performance-power efficiency" or "reduce power with near-peak performance" for scenarios where peak performance is not the only consideration.

6.4 Coordinated Management

The term "coordinated" has two connotations in our work. One sense of the word is coordinating among multiple techniques, to form a coherent approach to management. Another is coordinating among multiple constraints, such as performance floors and energy limits. We address both multiple mechanisms and multiple types of constraints.

We compared the effect of multiple management decisions applied individually and in a coordinated manner [30]. We developed microarchitectural simulation infrastructure to estimate performance, power, energy, and temperature throughout program execution. The processor model is based on a 90 nm version of the Alpha 21364 and is equipped with three independent management techniques:

- dynamic frequency and voltage scaling (DVFS) changes the operating speed

and total power for the full chip

- pipeline throttling restricts the rate of integer instruction issue to reduce dynamic power, and also eliminates leakage power by disabling a subcluster of the chip's datapath

- cache sleep mode reduces leakage power in caches while preserving data

We reported simulation results of the processor operating with no management, uncoordinated dynamic management, and empirically selected fixed management settings. We found that without management, the processor exceeded safety thresholds. Individual, un-coordinated mechanisms did prevent thermal-threshold violations, at the cost of prolonged execution time. Choosing *a priori* a coordinated combination of settings, fixed for the duration of the program execution avoided excessive pipeline stalls and safety violations, yet did not provide the best possible performance since it accommodated the worst case workload behavior, rather than tailoring the settings to *current* behavior dynamically at run-time.

A dynamic, online algorithm could converge to the best static settings for constant behavior, and for programs with time-varying behavior, a dynamic algorithm should be able adapt to find the best settings for each distinct epoch within the execution.

In our experiments with live hardware, we studied the two available techniques and found that one mechanism, DVFS, was superior, and the system did not require coordination of multiple techniques. We expect that coordinating multiple

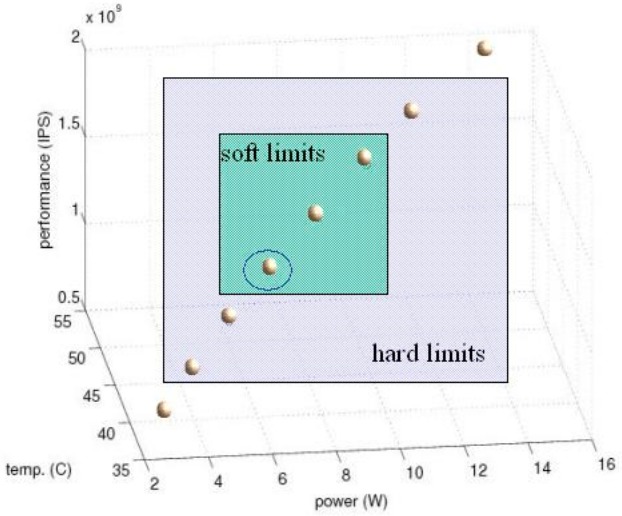


Figure 6.2: Goal specifications for 3 dimensions: hard limits, soft limits enclose valid volumes; objective function chooses 'best' point (circled).

mechanisms will be useful, if not mandatory, for both intra-chip and large-scale system control.

Coordinating between multiple dimensions is the other sense of coordinating, an important aspect of the PET approach. Figure 6.2 shows the goal specifications with a three-dimensional example of performance, power, and temperature. The hard and soft limits delineate volumes that correspond to strict bounds and preferences, respectively. The best p-state is chosen as the one corresponding to the point with the maximum benefit, as determined by the user-supplied objective function of

"minimum power." PET constructs a multi-dimensional space to form relationships between each dimension to provide coordinated control of multiple aspects of the system, rather than treating limits for each dimension as a separate stage of control.

6.5 Continuous Optimization

We believe the continuous optimization in the form of run-time adaptation will be essential to keep pace with environmental and workload behavior. Chapter 5 demonstrated how a drifting ambient temperature directly affected CPU temperatures. As noted in Chapter 3, workload intensity fluctuates over time, and the response to power-management varies by workload, as well. Chapter 3 also notes that system components behave as individuals, with nominally identical parts exhibiting unique power profiles. The variations in fabrication processes are expected to increase in future technology generations, increasing the spread in power and performance responses from individual components. The objective, too, will change over time as task priorities change and as components experience failure and upgrades. With many sources of component and workload variation, it will be critical for the manager to adapt to present run-time environment.

6.6 Implementations

Conceptually, the manager consists of three components: a sensor network to monitor chip activity; a command center that reads input, makes decisions, and gives directives; and a collection of actuators to turn the knobs. There are many ways to partition the manager within hardware and software components, and sev-

eral different sensor and actuator protocols [31]. Chapter 7 presents a prototype implementation in user-level software a Pentium M platform. This section discusses alternative implementations that we considered prior to our prototype development.

6.6.1 Interrupts on Host Processor

A single-processor implementation option would be a collection of dedicated registers and an interrupt protocol. The processor writes internal registers to set multiple thresholds for temperature, power, and energy. Hardware monitors would trigger interrupts when they detected threshold crossings. This approach is similar to a thermal assist unit [79] with programmable temperature thresholds, expanded to multiple monitoring parameters, with the option of multiple thresholds per parameter. Interrupts would keep the processor apprised of the changing status of the physical environment. The manager algorithm would be implemented as software executing on the processor in the operating system or user space. In a multi-threaded system, the manager could be a helper thread running intermittently under typical conditions and running as a dedicated single thread (after an interrupt alert) to handle emergency situations [54].

6.6.2 Service Processor

The register and interrupt protocol could also be applied in a multi-core system, in which multiple processors were controlled by a common manager. One option would be a small, low-power embedded core, similar to service processors found in large server systems, though co-located on-die with the primary cores. The embedded processor would require an interface with actuators to adjust the voltage,

frequency, and microarchitectural management settings. It would also require the addition of a specialized communication interface for collecting physical sensor information, such as temperature readings, and event indicators, such as cache miss rates and performance counters from the primary processor [7]. The manager's specialized software would execute on the service processor and control the chip via device drivers. This option would provide flexibility to change algorithms by loading new programs or invoke multiple algorithm options by calling a variety of subroutines. Software updates could keep pace with server system upgrades and management algorithm innovations [54]. Figure 6.3 illustrates a configuration with a 4-core TRIPS system. In this diagram the embedded core controls separate voltage and frequency islands for computation and on-chip memory [27].

6.6.3 Larger System

In a larger multi-processor system, PET would coordinate multiple chips, blades, and racks. A hierarchy of manager components on each chip, blade, and within the operating system would coordinate the workload and hardware. Lower levels in the hierarchy would make localized decisions, such as voltage settings, and propagate setting and sensor information to its parent level. Higher levels of the hierarchy accumulate global information to make decisions such as turning off a blade for cooling, and to inform lower levels of the current goal states [31].

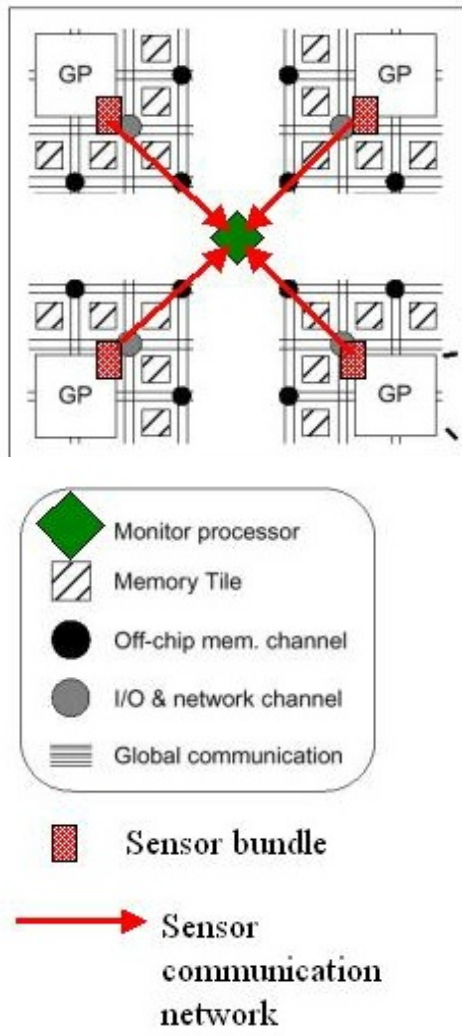


Figure 6.3: Multi-core PET scheme [27].

Chapter 7

PET Prototype

7.1 PET Prototype

To explore the key features of the PET approach, we built a prototype software implementation that controls a single-core Intel Pentium M processor via DVFS. The prototype platform offers a controlled environment with a low-power processor architecture that has been incorporated into systems from laptop computers to servers. In this section, we describe the prototype manager software and the method of selecting appropriate DVFS p-states to meet the specified goal. The prototype PET manager software first initializes event counters and software timers, establishes a connection via UDP to the measurement PC, raises a GPIO signal to indicate the beginning of a benchmark, then spawns the benchmark as a child process with the highest user-level priority.

During benchmark execution, the manager continuously loops through the sequence:

- **monitor** the system,
- **select** an appropriate p-state
- **activate** the selected p-state

Monitor
1. Read event counters from registers with RDPMC instruction 2. Read power data from network socket (UDP) 3. Query temperature sensor via SMBUS
Select
1. Estimate performance at all p-states 2. Estimate power at all p-states 3. Estimate temperature at all p-states 4. Mark valid p-states for hard limits 5. Mark valid p-states for soft limits 6. Find intersection of valid p-states from hard and soft limits 7. Apply objective function (benefit) 8. Select p-state with maximum benefit

Table 7.1: Prototype p-state monitoring and selection process.

to achieve the goal, as shown in Table 7.1. At the conclusion of the benchmark, the manager software lowers the GPIO signal, prints the stored trace data to a file, and exits.

7.1.1 Monitor

The prototype manager gathers data from multiple sources, querying the fan controller for temperature data, logging event counter data for performance updates with the `RDPMC` (read performance monitoring counter) instruction, and reading from a network socket for power measurements. At a nominal sampling rate of 100 samples per second, the effective sample interval length varies slightly, with most samples within 10-15 ms and a mean sample length of 13 ms. Power samples arrive in UDP packets at a rate of 100 samples per second; when the UDP buffer fills due to a sample rate mismatch, data in the buffer are overwritten. Temperature is sampled less often, at a user-controlled rate of 1 temperature sample per N performance

samples (50 temperature samples per second in these experiments), via a custom driver to the LM85 fan controller.

Measured data are stored in a buffer with one entry per sampling interval; each entry consists of multiple fields, including the timestamp counter, performance counters, p-state settings, fan speed, current goal specification, and several other diagnostic fields. Temperature and fan speed data are gathered less often than other data; the most recently queried temperature and fan speed values are repeated in each entry in the buffer between sensor-reading updates. Due to sensor delays and infrastructure limitations, power and thermal data do not correspond precisely to the timestamp with which they are stored in the buffer. The prototype treats each entry of the buffer as a collection of simultaneous measurements, ignoring the small time skew between measurement types.

7.1.2 Select

The PET prototype software predicts the outcome for performance, power, and temperature for each p-state based on measurements with the current p-state. We apply predictive models previously described to fill one N x M array for each dimension: power, performance, energy, and temperature. Preliminary experiments used 8 DVFS and 8 clock throttling settings, for 8x8 arrays for power and for performance. After pruning clock throttling from the study and adding temperature, the prototype uses three 8 x 1 arrays, one each for performance, power, and temperature. Energy is calculated but not used in the current prototype version.

First, the manager estimates performance in terms of instructions per sec-

ond, IPS, for each p-state and stores the results in an array, with one array element per p-state. It calculates estimated power for each p-state and stores the results in a separate array, and estimates and stores temperature for each p-state in a third array. The collection of arrays forms a multi-dimensional view of predicted behavior, as shown in Figure 7.1. The figure illustrates a snapshot for one measurement sample of PET prototype operation from the benchmark `gzip` executing at the 1200 MHz (unthrottled) p-state. After the PET manager software compiles estimates for three dimensions (performance, power, and temperature) for each p-state, it proceeds to evaluate each p-state's outcome relative to the goal.

Goal Specification: Initial goal parameters are specified on the manager command line and are adjusted via user software signals `SIGUSR1` and `SIGUSR2` during execution. Minimum and maximum bounds for power and temperature hard and soft limits are specified in Watts and degrees Celsius, respectively. Performance hard and soft limits are specified with a throughput metric, percent of maximum IPS. The IPS metric allows comparison for different management alternatives for a given workload, sufficient for our prototype. An ideal performance metric for a large-scale PET implementation would encapsulate both latency and throughput, to allow a more complete specification in systems that perform tasks such as database transactions, where latency may be included in a quality-of-service contract. Case statements within signal-handler functions select the action to apply to the user signals. In the case of experiments in this section, `SIGUSR1` increases the bounds for power, temperature, and performance, while `SIGUSR2` decreases the limits by

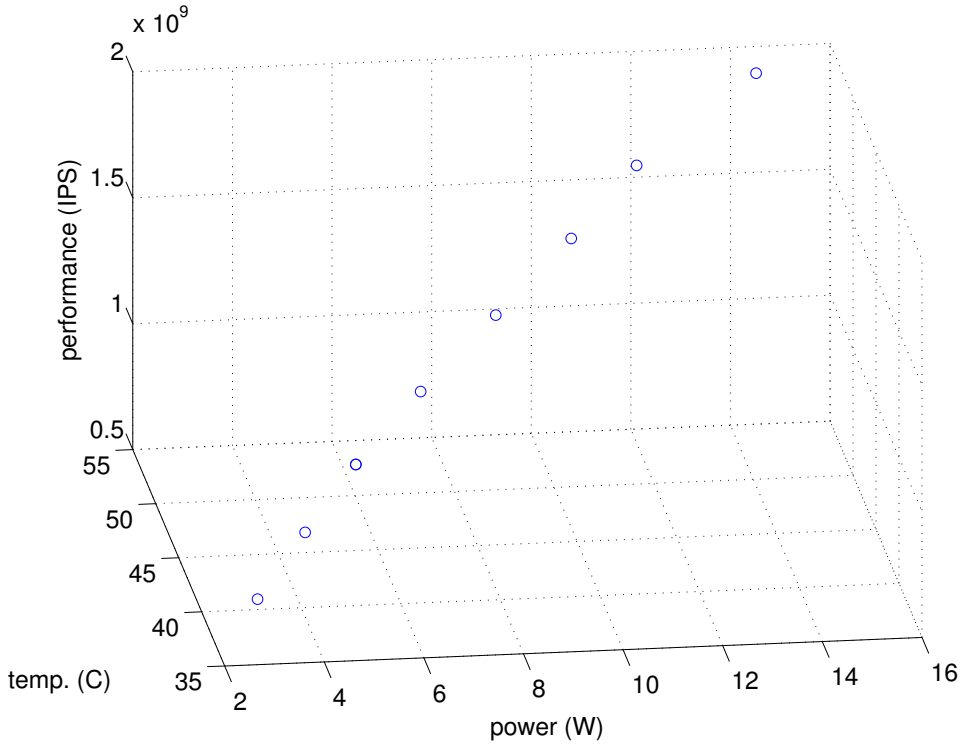


Figure 7.1: Multi-dimensional space: projected performance, power, and temperature for one sampling interval.

incremental amounts.

P-state Selection: The manager evaluates the p-state choices in three steps. First, the manager iterates through each array, comparing the estimated values to the hard limits. It marks each value that violates hard limits as invalid. In the second step, the manager iterates through the valid choices in each array, comparing estimated values to the soft limits. The manager marks values outside the soft limits with a separate soft-limit invalid marker. At this point, each p-state has been evaluated for both hard and soft limits, for all each dimension: performance, power, and temperature. In the third step, the manager must choose one p-state from the collection of possible p-states. The manager uses valid markers to identify the set of p-states that satisfy both hard and soft limits. For each of these valid candidates, the manager evaluates the objective function, and chooses the one with the maximum benefit as defined by the objective function. For instance, if the objective function is "minimum power," the manager chooses the p-state that meets hard and soft constraints with the lowest power.

The goal syntax allows a performance-oriented objective within a preference region of lower power, allowing the user to tailor the goals to accurately reflect the objectives. This approach is different than most prior art, which for a "maximum performance" objective would choose best performance at all costs. Point locations within the space will change over time due to workload activity, ambient temperature, etc. so the manger continuously repeats the sequence to tailor p-states to current conditions.

- In the case that no p-states meet hard limits, the manager chooses a predetermined safe mode p-state and emits a warning. The safe mode is presently programmed as 1000 MHz, unthrottled operation, which yields a reasonable power consumption without excessively degrading performance.

- Hard limits always have precedence over soft limits, effectively clipping soft limits to hard-limit boundaries by prioritizing the hard-limit invalid markers over soft-limit valid markers.

- If multiple p-states evaluate to the same benefit, they are equivalent under the selection choices and any of the p-states are acceptable. With floating-point resolution for benefit calculations, this case will be very rare in the prototype, but if it does occur, the code is structured such that the first p-state evaluated of the tie will be selected.

7.1.3 Activate

After a p-state has been selected, the PET prototype software calls the `cpufreq` driver to adjust the voltage and frequency settings to the new p-state. The processor stalls for up to 500 μs to change the voltage and frequency, a relatively small stall time compared to the 10-15 ms time interval of the prototype sampling interval.

7.2 Preliminary Power-Performance Experiments

Two of the key benefits of the PET approach are the flexibility to adapt the goal to different scenarios, and to adapt to variable workload behavior. In

this section, we present a series of experiments in which we explore the use of the PET approach for gracefully responding to sudden shortage in power supply and for meeting a series of distinct goal targets. These experiments were performed with early versions of the PET software, including preliminary power models and no thermal sensors. We include these studies to demonstrate the flexibility that the goal specifications provide to the PET manager, and share observations and insights we learned throughout the prototype development.

The series of experiments demonstrates two important capabilities in the PET prototype. First, for a given workload, the prototype selects appropriate p-states to meet the goal target. Second, for a given goal target, the prototype dynamically adjusts to different p-state settings based on the current workload behavior.

7.2.1 Case Study: Power Emergency

PET can be used to improve system stability by providing goal-driven reactions to dramatic changes in operating conditions. Figure 7.2 shows the PET manager responding to a sudden change in the hard limit for power while the `perlbmk` benchmark executes. Such a change may be the result of a failure of a redundant power supply, chiller failure in a machine room, or other potentially disastrous situation. In this experiment, the power-management setting is initially 1.8 GHz and unthrottled, and the system is running the Windows XP operating system. A software signal arriving at approximately 12 seconds on the timeline indicates a new power limit. A signal handler in the manager changes the goal to reflect a new hard limit of 5 Watts, then the manager predicts power and performance for the 64 com-

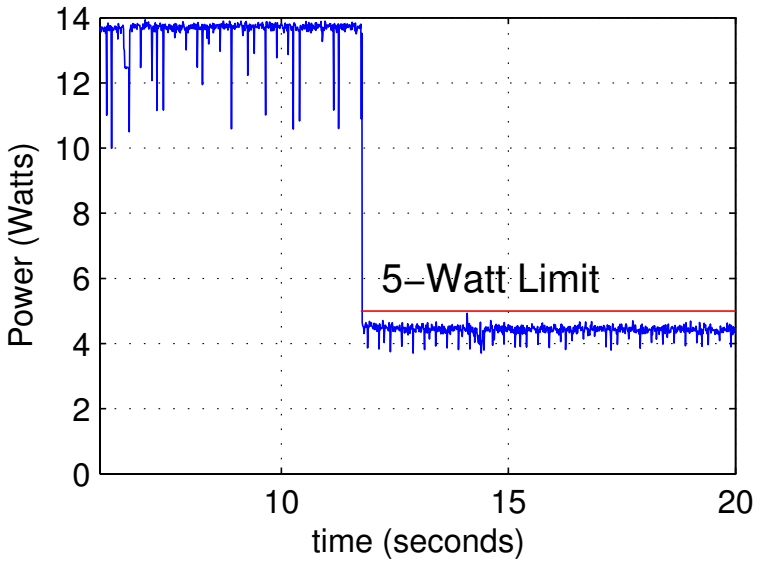


Figure 7.2: Sudden drop to 5-Watt power limit for `perlbmk`.

binations of DVFS and clock throttling levels. It selects and actuates the 800 MHz setting with the best performance that meets the new limit. The average power at the new setting is 4.42 W, with a range from 3.53 to 4.74 W, resulting in a small amount of headroom within the limit for this benchmark. With finer control, such as frequency increments of 100 MHz rather than 200 MHz, the headroom could be reduced to improve performance within the power limit.

The effectiveness of managing a power crisis is limited by response time and the power model accuracy. The response time in these experiments is instantaneous on a 10-ms scale. The power model provides sufficient accuracy to determine a new frequency to meet the suddenly reduced power limit, without over-reacting and degrading performance with a more conservative low-frequency p-state.

7.2.2 Case Study: Power Budgets and Performance Floors

A goal-oriented approach allows a power manager to tune settings for the current workload and conditions, rather than using pre-determined responses that may not be appropriate for the current situation. The following examples illustrate the PET manager's operation for three typical goal scenarios.

Scenario 1: Maximum performance to meet a deadline, provide superior performance to high-priority workloads, or to complete a job quickly and turn off components with high standby power. The soft limits specify high expectations for performance and power, within safe power limits and minimal performance hard limits.

Scenario 2: Mid-range power and performance for use with parallel workloads distributed on multiple cores that maintain throughput while running each core at a mid-range core frequency and supply voltage to save power. The goal specification for scenario 2 employs the same hard limits as scenario 1, with soft limits that constrain operation to a lower power budget. The objective function specifies maximum performance; the manager will find the best possible performance point within the hard and soft limits.

Scenario 3: Minimize energy to control operating costs in a data center. Hard limits enforce performance constraints to meet quality-of-service contract requirements; soft limits are equal to the hard limits. The objective function specifies minimum energy, which in the prototype is equivalent to minimizing power (due to the low leakage power).

	Performance			Power			
	Hard Limit	Soft Limit		Hard Limit	Soft Limit		Objective
Scenario	Min	Min	Max	Max	Min	Max	
1) high performance	30%	90%	100%	17.0 W	0.0	16.0 W	maximize performance
2) mid range	30%	60%	100%	17.0 W	0.0	11.0 W	maximize performance
3) low energy	70%	0 %	100%	17.0 W	0.0	20.0 W	minimize energy

Table 7.2: Goal specifications for preliminary experiments.

Table 7.2 lists the goal specifications for each scenario. Figures 7.3 through 7.5 show the manager-selected frequency and resulting power and latency for SPEC CPU2000 integer benchmarks `gcc` and `perlbmk`. Each data point in the graph represents one sampling interval, approximately 10-15 ms with the Windows XP operating system. Data for each scenario are plotted on the same scale per benchmark to show the trade-offs between shorter execution time and lower power consumption.

The prototype's simple power and performance estimation models lead to constraint violations for both `gcc` and `perlbmk`. In scenarios 1 and 2, the manager under-estimates power and chooses frequencies that overshoot the soft limits of 16 Watts and 11 Watts, respectively. In scenario 3, the manager chooses 600 MHz for all time samples of `perlbmk` and the majority of `gcc` sampling intervals. The manager over-estimates performance in these cases, choosing a frequency that is sufficient according to its estimate but that does not meet performance requirements in operation. The over-optimistic prediction was based on treating these benchmarks as heavily memory-bound, which reduces performance sensitivity to frequency. Refer to the `swim` benchmark in Appendix A to view a memory-bound case where the

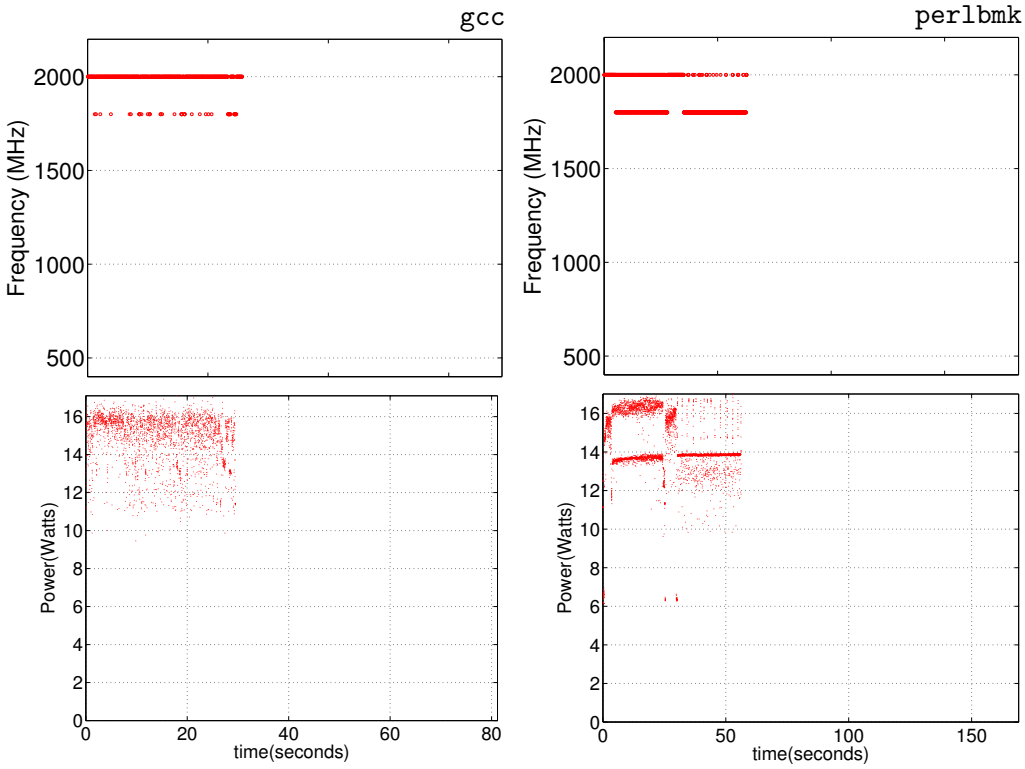


Figure 7.3: Scenario 1: `gcc` and `perlbmk` execute at high frequencies, 2000 MHz and 1800 MHz.

600 MHz p-state would be sufficient to achieve better than 70% of IPS_{max}.

With these experiments during prototype development, we gained insight into the prototype's sensitivity to power and performance estimation accuracy. As a separate issue from the optimistic performance estimator, we also observed the detrimental effects of using measured power in a feedback loop without information about the p-state under which the power was recorded. In this case, a power model that served well for steady microbenchmarks was inadequate under typical conditions of variable workload behavior and fluctuating p-states, as the model was

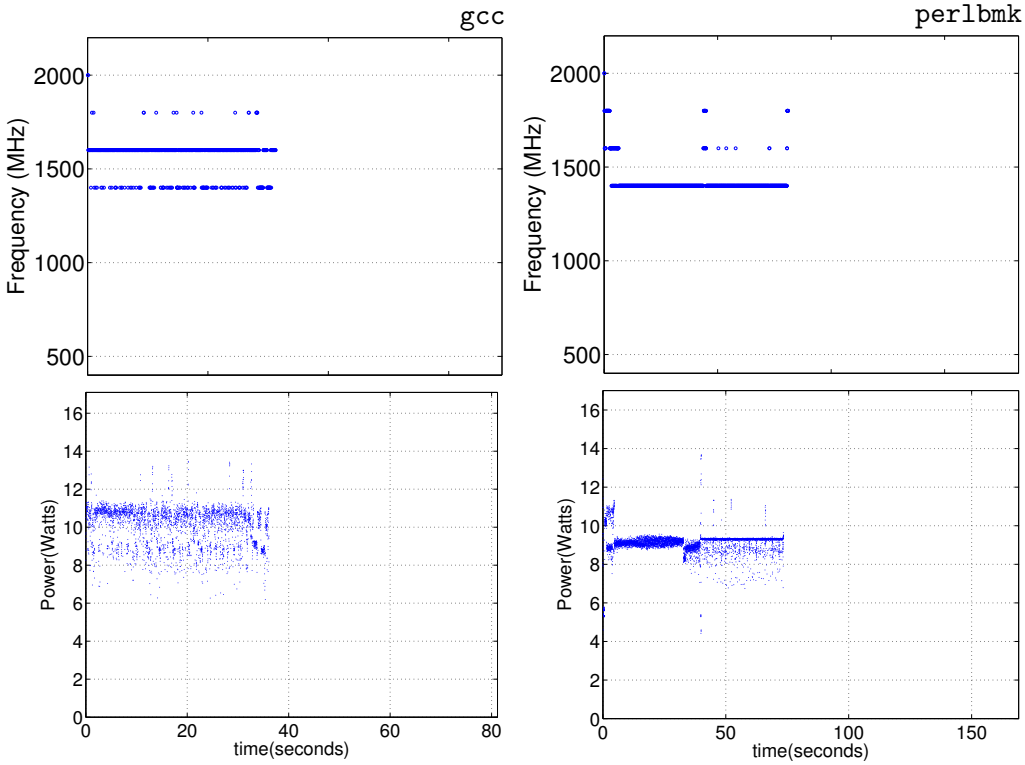


Figure 7.4: Scenario 2: `gcc` and `perlbmk` operate between 1400 MHz and 1800 MHz; `gcc` spends more time at higher frequencies than `perlbmk`.

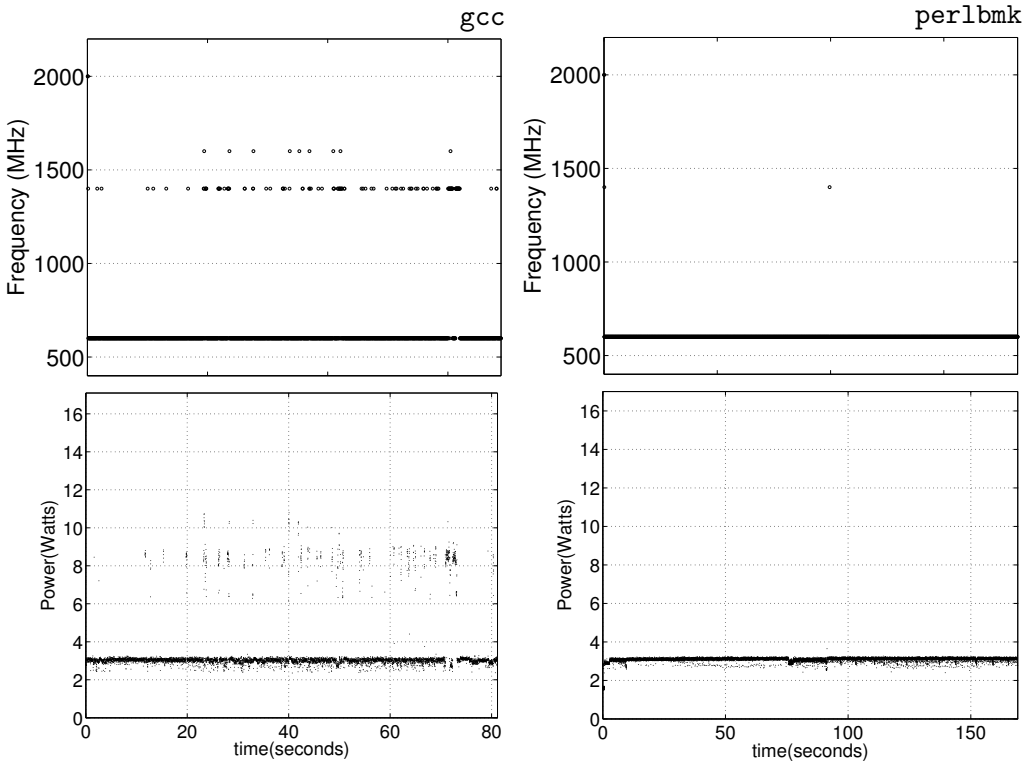


Figure 7.5: Scenario 3: `gcc` and `perlbmk` operate at low frequencies, typically 600 MHz. The manager chooses frequencies at 1400 MHz and 1600 MHz for some samples of `gcc`.

unable to distinguish the effects on power caused by workload activity or p-states, and thus could not tailor future p-state selections to workload characteristics. Even with only eight useful p-states to choose from, it was possible–and even likely–to make a poor p-state selection when the performance and power estimates did not reflect current behavior. Subsequent revisions to the manager software improved the prediction accuracy, which in turn improved the ability to evaluate the PET approach with the prototype.

7.2.3 Case Study: Balanced Power-Performance Goal

We migrated from the Windows XP operating system to Red Hat Linux, corrected the memory-boundedness classification issue demonstrated in the previous experiments, and upgraded the power models.

We evaluated a multi-dimensional optimization of **balanced power and performance**. In this scenario, the power limits are 20 Watts and 12 Watts for hard and soft power limits, respectively. Performance hard limits are set to zero (unconstrained) and performance soft limits are set to a minimum of 70% IPS_{max}. The manager must find the subset of points that meet both maximum-power and minimum-performance constraints simultaneously. The objective function balances performance and power by minimizing the distance from the origin in normalized power-performance space. The function is stated as $minimize\sqrt{p^2 + l^2}$, where p is normalized power and l is normalized latency.

Performance is lower than at maximum frequency, and would be useful in situations with parallel tasks distributed on multiple cores to match rate and through-

put for energy efficiency, or for meeting–without exceeding–performance service level agreements. It is also useful for saving power in components with a high overhead for powering down.

Figure 7.6 illustrates the full SPEC CPU2000 suite executing at a fixed 2 GHz p-state and with the balanced power-performance goal. Table 7.3 lists the performance, power, and energy metrics for the balanced goal compared to the unconstrained operation. The balanced-goal execution is substantially slower, about 98 minutes compared to about 73 minutes for unconstrained operation, and uses substantially less energy, approximately 42 kJ compared to about 58 kJ.

The primary benefit of this experiment was testing the prototype manager's capability to evaluate a multi-dimensional objective function. The prototype p-state selection process begins with finding the intersection of points that meet hard and soft limits, and then applies the objective function to choose among valid points. Previous experiments focused on the hard and soft limits, with simple uni-dimensional objective functions of "maximize performance" or "minimize power." The balanced power-performance experiment created a more complex objective function that relates the power dimension and performance dimension, explicitly defining the benefit across both dimensions. The multi-dimensional capability of the objective function is an important feature of the PET management approach.

7.3 Prototype Demonstration

Following the previous experiments with power and performance, we upgraded the power model based on power feedback and we added thermal monitoring

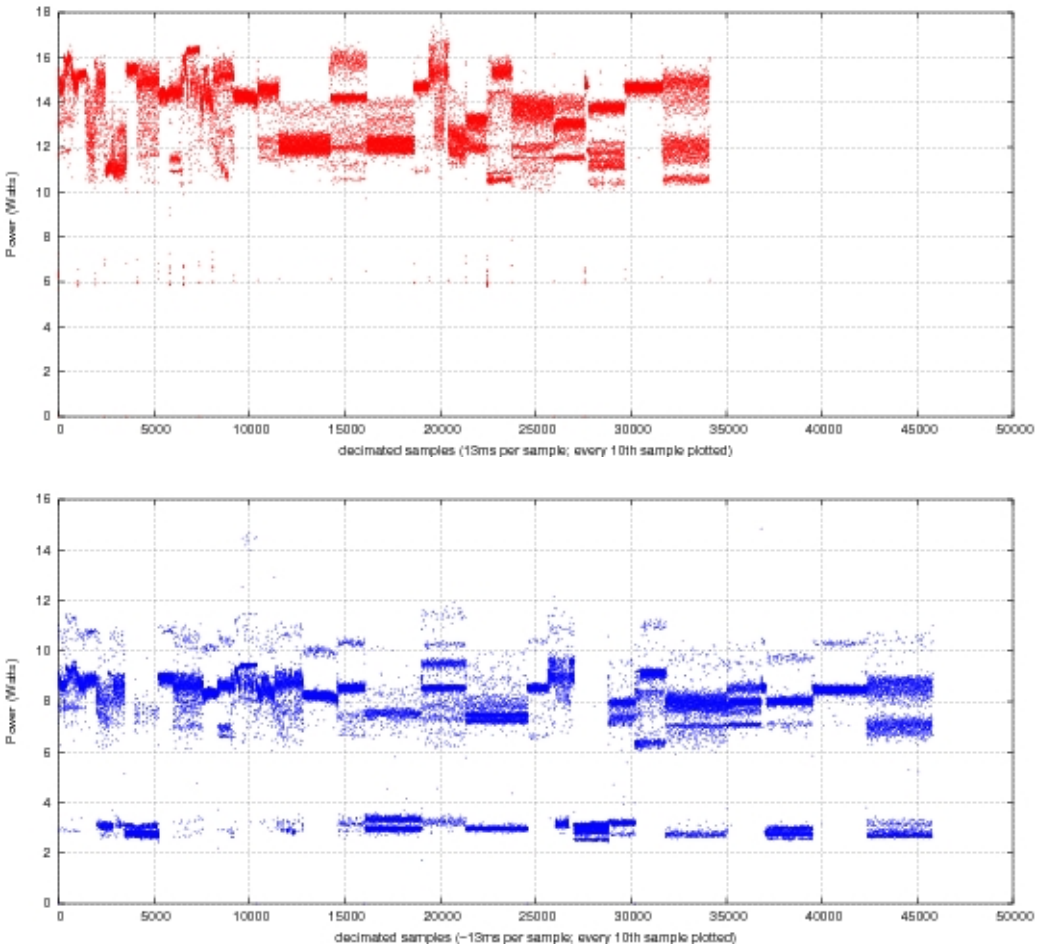


Figure 7.6: SPEC CPU2000 suite for fixed 2 GHz p-state (unconstrained operation) and for a balanced power-performance goal.

Scenario	Execution Time minutes	Max Power Watts	Mean Power Watts	Total Energy kilojoules
continuous 2 GHz operation	72.7	17.38	13.37	58.3
balanced performance and power	98.4	15.43	7.05	41.6

Table 7.3: SPEC CPU2000 balanced power and performance summary.

and prediction. In this section, we use our full prototype to illustrate how the PET approach guides the system to a desired outcome by adapting p-state choices in response to variable run-time environment and workload behavior. In the first experiment, the manager chooses appropriate p-states to provide the best performance within fixed power and thermal limits. In the second experiment, we illustrate a scenario where the goal changes as a user moves the desired operating point along a spectrum of performance to cost trade-offs.

The workload is the `ref` input set of the SPEC CPU2000 suite, consisting of 11 integer and 15 floating-point applications. The SPEC CPU2000 suite completes in approximately 70 minutes with a continuous frequency of 2 GHz on our Pentium M system, with an average power consumption of 13 Watts.

7.3.1 Fixed Goal, Variable Workload

We tested the prototype with a PET manager target of achieving the best possible performance within a power and thermal budget. The goal conditions in this experiment represent the case of a well-provisioned system that is directed to mid-range operation to conserve power to reduce operating costs. Hard limits delineate strict bounds of 17 Watts and 70 °C. The hard limits are generous, with greater power consumption allowed than the workloads typically demand, and a high temperature limit for this system. Soft limits impose additional preferences for a 10-Watt power budget and 43 °C thermal limit. The objective function is "maximize performance." Although the objective function commands performance, the soft limits indicate a preference to consume less power than the workloads would demand

at full speed; the combination of the objective function and soft limits express the compromise between performance and power. The temperature limit of 43 °C for this system may or may not limit frequency to curb heat output, depending on the ambient conditions.

Graphs in Figure 7.7 through 7.9 indicate how the manager directs frequency changes in response to varying workload demands to meet the fixed goal specification. The CPU frequency hovers between 1200 and 1400 MHz with excursions up to the maximum 2000 MHz and down to 1000 MHz, tracking the workload intensity.

Figure 7.8 illustrates the power profile; most measurements are recorded between 5-10 Watts, with occasional lower- and higher-power points. The unpredictable timing of the power feedback by UDP in this experimental system causes outdated information from periods of previous low activity to adversely affect frequency choices, causing higher power than the budget would allow for the *current* activity. Even with unreliable power information, the manager rarely violates hard power constraints in these experiments, overshooting the hard constraints for a total of 91 milliseconds out of 99.1 minutes of execution. The manager respects soft power limits for most decisions, violating the soft limits for 0.52% of the execution time with an average of 1 Watt, a 10% overshoot. The temperature plot in Figure 7.9 shows that measured temperatures were well within a generous 70 °C hard limit (not shown on this scale). The CPU temperature occasionally breached the 43 °C soft-limit, 0.5% of the time with a mean of 1.1 °C overshoot.

While the PET manager intends to respect all constraints, run-time system behavior does not always match the manager's expectations. Using previously

measured data as an indication of future response poses a fundamental problem for controlling variable system behavior, whether by predictive or reactive processes. In addition, the prototype PET manager relies on low-overhead estimates of power, performance, temperature, etc. based on sensor data recorded for one p-state and extrapolated to all other p-states. Estimation error in each prediction increases the risk of constraint violations. However, permitting occasional mis-steps—with the capability to correct overshoots quickly—allows the system to operate near constraint boundaries without being hampered by more conservative margins.

Overall, the manager chooses appropriate p-states on the fly to meet the desired goal with variable workload activity in most cases. Mild constraint violations such as those observed in this experiment may be acceptable in most situations. Systems that do require tighter control could add additional safety measures, including conservative guard bands on the hard constraints, more detailed estimation models, or upgrades in monitoring hardware (such as integrated power monitoring features available on IBM blade servers).

We demonstrated that the PET manager prototype is feasible and produces the desired results with very good accuracy, despite delayed sensor information and fluctuating workload characteristics. The goal specification and continuous optimization enables a PET manager to adapt to run-time conditions and maintain desired system operation.

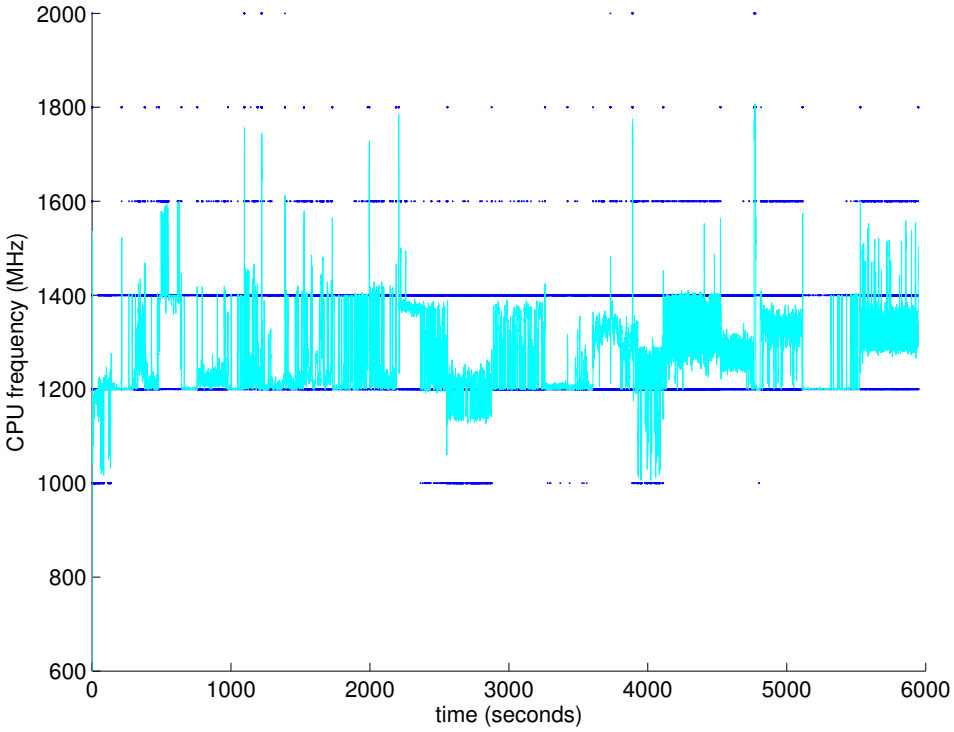


Figure 7.7: Fluctuating CPU frequency (dark blue) and 1-second average (cyan) during PET prototype experiment with fixed goal and variable workload.

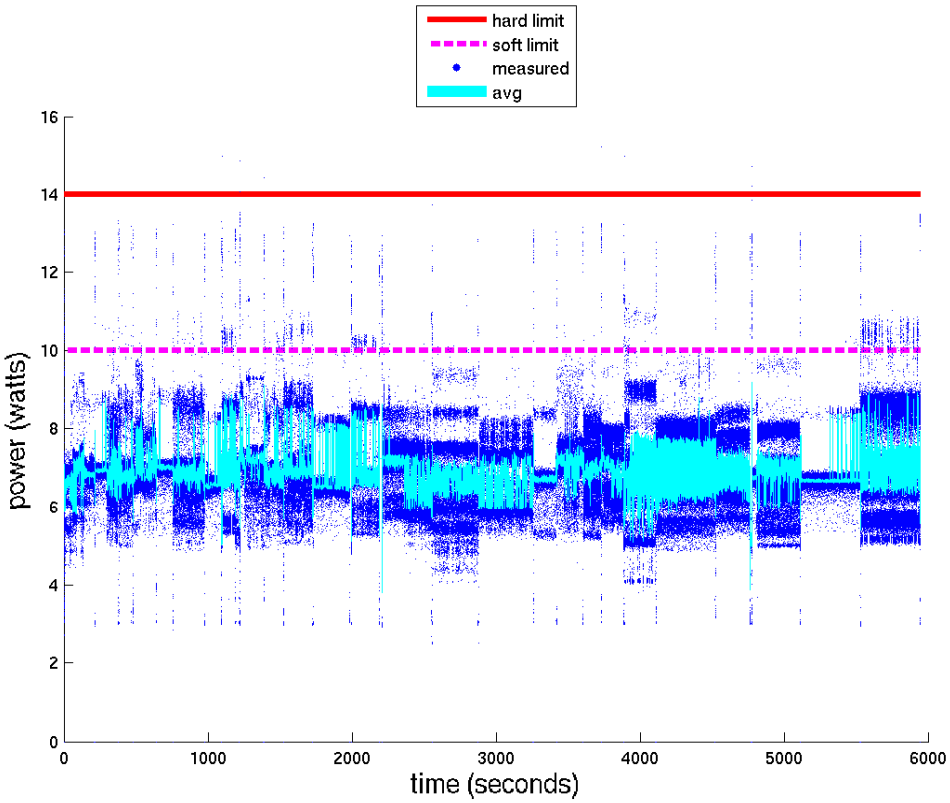


Figure 7.8: Measured and 1-second average power during PET prototype experiment with fixed goal and variable workload.

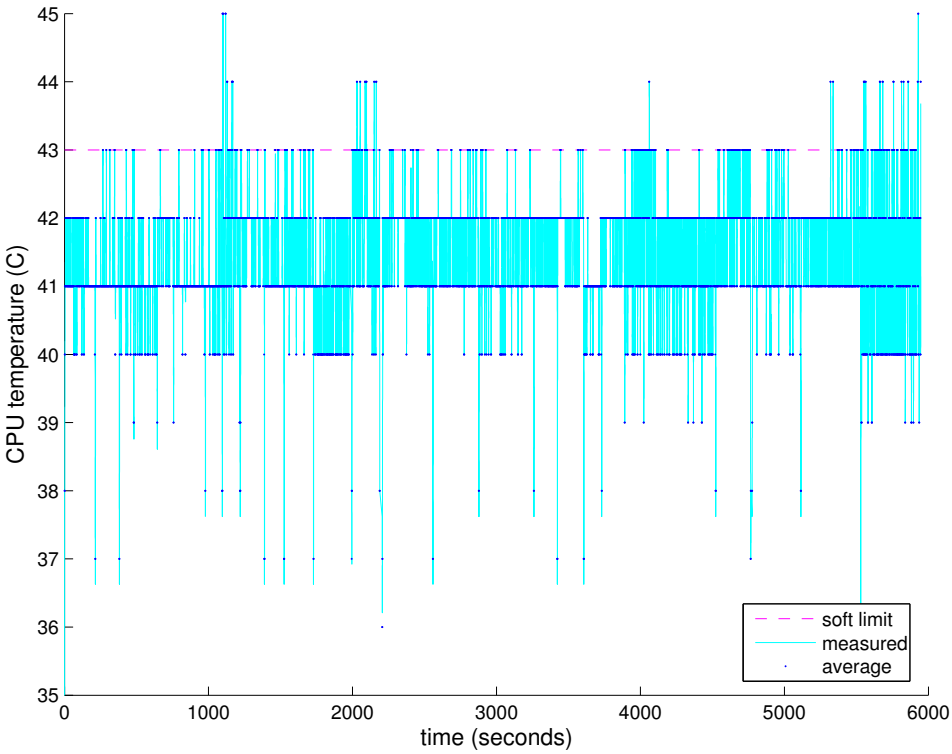


Figure 7.9: Measured CPU temperature (dark blue) and 1-second average (cyan) during PET prototype experiment with fixed goal and variable workload.

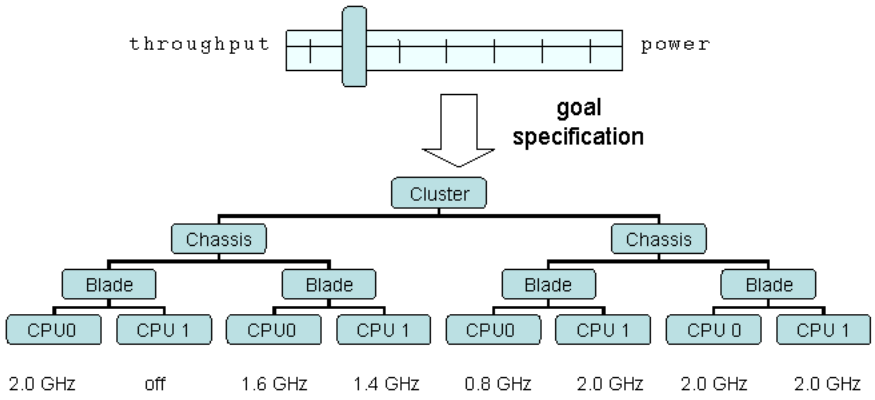


Figure 7.10: Cost-performance spectrum goals translated to DVFS settings in multi-core system.

7.3.2 Variable Goal and Workload

The next challenge is to demonstrate that the manager is capable of tracking variable workload behavior while the goal also shifts over time. One important consideration for power and thermal management is that the objectives and system response will change through time, and that a one-size-fits-all approach will not suffice. We create a scenario of a data center, with high-level objectives of meeting throughput quotas while managing operating costs. In this scenario, there is no advantage to executing faster than necessary, and there is a high penalty for exceeding the electricity budget or failing to meet performance requirements. The cost-performance priorities shift throughout time due to task urgency, system load, and other factors. Conceptually, the experiment emulates an operator moving a sliding bar between two extremes, high performance and lower operating cost, as shown in Figure 7.10.

The PET manager uses the goal specification to translate the high-level objectives into measurable quantities. The objective function is set to "maximize performance" and hard limits are fixed at maximum temperature of 80 °C maximum power of 17 Watts, and minimum performance of 50%. Performance requirements specify the minimum acceptable instructions per second (IPS), as a percentage of IPS expected at the maximum frequency. In this experiment, the objective function and hard limits are fixed; soft limits that define the preferred region of operation shift as the user expresses priorities between better throughput and lower operating costs. Initial conditions for the soft limits are set to 75 °C, 14 Watts, and 80% of max IPS. In this prototype system without a graphical interface, user actions are conveyed by

a bash shell script that sends signals to the manager's process to modify the soft limits, nudging the soft-limit boundary one notch either toward better performance with SIGUSR1 or toward lower cost with SIGUSR2 every 4 seconds. The arbitrary pattern of SIGUSR1 and SIGUSR2 signals mimics the change in objectives over time by an external operator, independent of application activity. A function handler in the prototype software provides the qualitative to quantitative mapping, translating each signal received as an incremental adjustment in each dimension simultaneously: +/-10% performance, +/-5 °C, and +/-1 Watt. SIGUSR1 increases limits and SIGUSR2 decreases limits. As the soft limits shift in response to the user preference, they may exceed hard limits, in which case the manager overrides user preferences and clips the soft region boundaries to the hard limit values.

Graphs in Figure 7.11 demonstrate the response to change in soft constraints for each dimension. The frequency plot illustrates the fluctuating frequency levels chosen to meet the moving target and workload behavior. The frequency ranges from 1000 MHz at the more restrictive power budgets to 2000 MHz for regions of higher performance expectations and/or workload behavior that could meet the power budgets at higher frequencies. The first plot shows performance as IPS. The hard limit is 50% of max IPS. The soft limit slides higher or lower every few seconds; the measured performance should be equal or greater to this minimum performance preference. The power plot illustrates the soft power limit adjusting in concert with the performance limit. As the user indicates higher performance, the power budget is relaxed; for lower operating costs, the power budget becomes tighter. The thermal plot also shows the hard and soft limits, although in this experiment, temperature

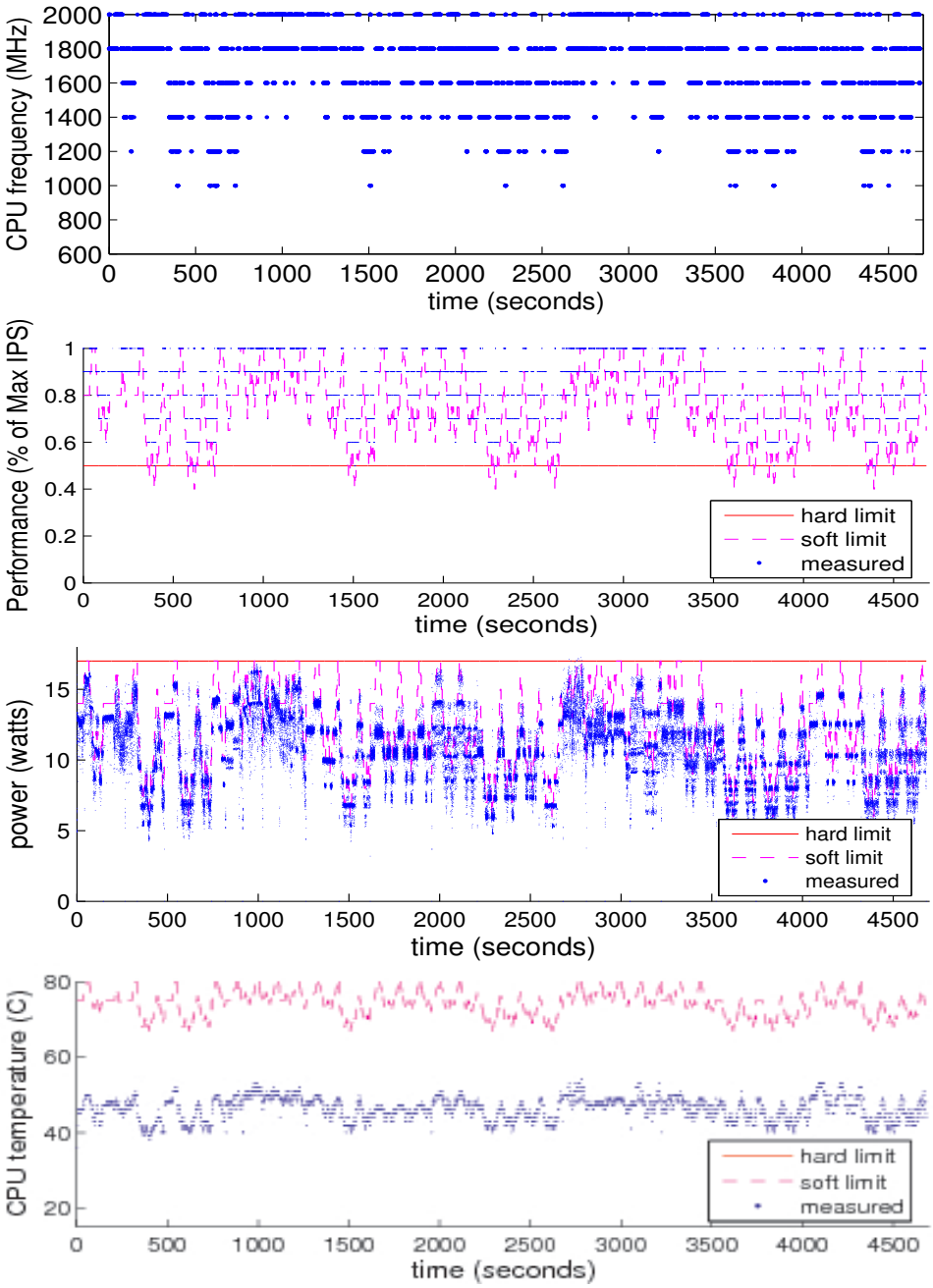


Figure 7.11: Variable cost-performance goals.

was not a limiting factor, as evidenced by the slack between the limits and measured values.

The manager meets the performance hard requirement and maintains performance near the soft limits, violating the soft limits 0.3% of execution time with a slight undershoot of 0.06% on average. The manager violates power hard constraints in 0.01% of execution time and power soft limits for 12.5% of execution time. The manager does not violate hard or soft temperature constraints in this experiment. The greater number of violations in this experiment is due primarily to exposing regions of lower fidelity in the estimation models as the system moves through a wider range of operation as the goal shifts from higher performance through lower-cost targets. It is important to note that despite the limited accuracy afforded by imprecise models, the manager maintains control of the system due to continuous monitoring that provides opportunities to re-evaluate p-state choices often, every 10-15 ms.

7.4 Summary and Discussion

We illustrated the key features of the PET approach with a prototype implementation that manages a Pentium M system with user-level software. The prototype platform provided a detailed view of power and performance under realistic workload and environmental conditions. As a test vehicle, it offered a means to investigate the following questions concerning a PET approach:

1. Is management overhead reasonable?

Yes. Even with a software timer interrupting the program every 10-15 ms for calculations, the prototype manager overhead is negligible. We executed the SPEC CPU2000 suite at 2GHz with and without the prototype manager, using the Windows XP operating system, which had a larger timer variation than the Linux operating system. We observed a 4% increase in run time, compared to run-to-run variation over twenty executions of the full benchmark suite of 2%.

2. Does the goal expression convey sufficient information?

   Yes. The three-part goal syntax of hard limits, soft limits, and objective function allows full expression to direct the chip to the desired operating region, and reflect the user's preferences. We observed power and performance distinctions between goals that share a common hard limit; the differences in the soft limits and objective function lead to distinct decisions for p-state choices to reflect user preferences.

3. Can a chip-level manager with limited information detect and exploit transient, fluctuating behavior to manage power and performance close to—without exceeding–the limits?

   The prototype manager generally chose settings well even with a less-accurate power model, but occasionally chose poor settings, as evidenced by power exceeding 15 Watts despite a 12-W limit in one experiment. As Figure 3.4 in Chapter 3 showed, the assumption implicit in the prototype estimation models that recent history can predict future results is valid most of the time,

but is not universally true. Accommodating varying hardware and software behavior will be essential to push the constraints envelope safely. Despite *a priori* power models that rely on recent history, the prototype manager was able to maneuver the operating states near the stated goals for a vast majority of time samples; based on experience with the prototype, we believe that a PET manager could harvest useful, timely data from the processor core and apply it wisely.

We demonstrated that the PET approach continuously adapts to workload behavior as it strives to meet target operating points in a series of scenarios. Early explorations with preliminary versions of the prototype for performance, power, and energy suggested that the PET approach was feasible and showed the tradeoffs between power and performance on the Pentium M platform. First, we demonstrated the capability to gracefully transition to a low power limit, simulating an emergency event for the power supply. The PET prototype was able to choose a suitable frequency for the `perlbmk` benchmark to maintain the 5-Watt limit. We also explored a range of objectives with two SPEC CPU2000 benchmarks, `gcc` and `perlbmk`, then we executed the full SPEC CPU2000 suite with a multi-dimensional goal.

We further refined the prototype with temperature sensors and the most effective p-state for a full prototype demonstration in two situations. First, we presented results that show PET maximizes performance within power and thermal budgets by adapting DVFS settings to dynamic workload behavior with the SPEC CPU2000 suite with the `ref` input set. In this case, the prototype manager generally adheres to hard constraints throughout execution and overshoots soft limits with

minor excursions. In the second set of experiments, both the workload behavior and the goal modulate during execution. In this scenario, the manager violates soft limits more often as it handles a wider range of operation; in the worst case, it violates power soft limits for 12.5% of execution time, though it maintains performance within the preferred range 99.7% of the time.

The prototype goal-driven controller was able to interpret the goal specifications into DVFS p-states continually on a feasible time scale with negligible overhead, choosing reasonable settings in most cases. The level of indirection afforded by the goal specification and the level of detailed information from performance event counters together provide a solution to specify true preferences at a high level, with the capability to enforce them at a lower level.

The prototype demonstrated the feasibility of the PET approach with a single-processor system. The prototyped approach holds promise for large-scale systems with a hierarchy of PET managers that each control a local region and together manage a larger system. In a cluster-level PET manager, goal specifications could be determined by human operators or automatically updated by operational utilities.

We analyzed traces collected on the prototype platform to investigate the sensitivity of power measurements to sampling interval length. We demonstrated that longer samples attenuated the power variation. Controlling power on a longer time scale will be simpler than on a short time scale due to the similarity among power values for longer sample lengths but longer time scales obscure short excursions beyond limits, which may have negative side effects for power supplies and

other physical components.

The prototype PET manager is limited to capabilities present in the Pentium M platform. We supplement the prototype experiments with simulations, and note that future systems are likely to provide elements lacking in the Pentium M system prototype.

First, we were unable to precisely duplicate our previous microarchitectural simulation investigations for orchestrating multiple mechanisms of pipeline throttling, sleep transistors for reducing leakage current, and DVFS [30]. We did evaluate two mechanisms in the prototype system with DVFS and clock throttling; however, we found that the manager only used the clock throttling option in the rare case that the goal specified bare-minimum power and tolerated the accompanying 27x slowdown in execution time. In typical operation, only unthrottled DVFS p-states were engaged. We also explored the use of DVFS and fan speed in an attempt to evaluate two orthogonal mechanisms, and found that fan speed held only a weak influence in this open-box system. There was a dramatic difference between the fan disabled and the fan spinning, yet gradations in rpm (revolutions per minute) had very little effect.

One outcome of our hardware experiments was confirming that DVFS was indeed a useful mechanism, far more effective than other presently available alternatives. However, we were not able to evaluate the prototype manager's ability to orchestrate multiple useful mechanism options.

Second, the Pentium M used in our experiments is a single-core processor. We developed a trace-based simulator in MATLAB to simulate multiple cores with

DVFS using data collected on the Pentium M system; however, the evaluation is limited to power, energy, and performance. Temperatures are recorded in the trace files but do not reflect thermal interaction that would be present in a chip multi-core system. Also, the individual traces do not reflect characteristics of a shared-memory configuration of multiple cores.

Third, direct measurements of power and temperature are limited in the Pentium M system. A single CPU temperature sensor resides in the processor package, and the motherboard hosts a second sensor. Both sensors have a fairly coarse resolution of 1 °C. A more complete picture of CPU temperature is available on other processors, such as the IBM POWER4 with 24 individual temperature sensors throughout the processor die.

Direct power measurements are not available within the Pentium M system. We formed a feedback loop through a separate computer instrumented with a data acquisition system; the resulting power data are delayed and lossy, which negatively affected power estimation accuracy in the prototype. IBM Blade servers currently provide direct power measurement on the motherboard; given the importance of controlling power, we expect future systems to include easily accessible direct power measurements.

Fourth, we envisioned extensive use of control-theoretic techniques in the PET approach [31]. Other researchers have demonstrated the potential for robust closed-loop control in computing systems [36] and power and thermal management [51], [81], [88], and we believe that a PET manager implementation would benefit from control-theoretic techniques, as well.

Finally, we focused on the processor-level PET manager in the prototype. The manager modules are composable, and the approach would naturally extend to a collection of interacting managers, whether in a distributed network, tree, or other configuration.

Chapter 8

Conclusion

A fundamental problem with power and thermal management is the difficulty of reducing power and heat output without sacrificing performance, which creates a complex web of inter-related constraints and requirements. Meeting these multiple, potentially conflicting objectives simultaneously is a difficult challenge, exacerbated by shifting environmental conditions and variable workloads, yet essential for future generations of high-performance systems.

8.1 Summary

In this dissertation, we have examined power and thermal management in microprocessors with investigations of power consumption and management in simulation, proceeding to a power and thermal characterization of an Intel Pentium M platform, and then applying our observations to develop a prototype PET manager to control performance, power, and temperature in a real system according to user-specified goals.

8.1.1 Power and Energy Analysis

In a study of the Alpha 21264 pipeline, we measured both the useful energy and the wasted effort due to mis-speculation and over-provisioning to identify

opportunities for power savings and energy efficiency throughout microprocessor pipelines. In another study, we compared three techniques for reducing static energy consumption in caches. One technique, dual-V_T, uses low-leakage transistors in the memory cell and faster, leakier transistors in supporting circuitry. A second technique, Gated-V_{DD}, interrupts current flow between the memory cell and voltage supply or electrical ground, discarding stored data in the memory cell. A third alternative is dynamic threshold modulation, MTCMOS, that places memory cells in a standby state via a back-gate bias, which preserves cell contents. Each leakage-control technique in the study effectively curbed leakage current with varying degrees of performance degradation. We found that state-preserving techniques were appropriate for the secondary cache due to the large penalty for re-acquiring data from off-chip memory locations. We also found that leakage-control mechanisms that increase instruction cache access time could actually cause more energy expenditure due to longer program execution times, degrading both energy and power. Thus, it is important to choose management techniques well to match the needs and requirements of the system.

Variation inherent in real systems poses a challenge to effectively control the system, due to shifting workload characteristics, complex interactions of management techniques, manufacturing variability, and other factors. Dynamic run-time management tailored to present conditions will be more effective than an *a priori* policy that does not guarantee the desired response.

8.1.2 Coordinated, Goal-oriented Management

We introduced a comprehensive, goal-oriented management approach that sorts priorities and balances conflicting goals, named **PET** for performance, power, energy, and temperature. A hierarchy of intelligence gathering and processing components in the manager would distribute decisions according to required response time: quick response for phenomena with shorter time constants, such as current spikes in the power distribution network, and longer intervals between decisions for slow-moving trends like gradual chip warming. With coordinated information from multiple sources and a goal-driven algorithm, the manager would adapt to the system environment and push the operating conditions to the edge of acceptable limits [31].

The PET approach provides a level of indirection between macro objectives, such as reducing operating cost or delivering high performance, and micro directives, including DVFS settings and other power-management choices. Goal-driven decisions reflect relevant run-time conditions, rather than pre-defined policies, and a concise specification for desired outcome provides an opportunity to customize operation to conserve or spend power resources as situations warrant, delivering performance on demand.

Characterization To explore the PET approach, we performed a series of studies with a prototype implementation in user-level software executing on a Pentium M machine instrumented with a highly accurate power monitoring system. First, we characterized the Pentium M system response to workloads, power management

mechanisms, and cooling systems to identify opportunities for power and thermal management. In one study, we characterized application response to two power management techniques, DVFS and clock throttling, applied individually and in combination with a set of microbenchmarks designed to capture core and memory subsystem activity. We analyzed microbenchmark power and performance for three dataset sizes corresponding to core-bound (level 1 cache), intermediate (level 2 cache), and memory-bound programs. We found that for techniques applied individually, DVFS is more effective than clock throttling at reducing power while preserving performance. With both techniques applied together in each of the 64 combinations of eight frequencies and eight clock throttling levels, we demonstrated that a performance or power target can be achieved with multiple combinations of DVFS and clock throttling settings; however, DVFS provides superior power-performance efficiency and takes advantage of fewer CPU cycles per memory accesses.

In another study, we identified the thermal response of the instrumented Pentium M system, exposing several useful properties to exploit in power and thermal management decisions. We identified the two-stage thermal response to p-state changes: a quick initial change (milliseconds) followed by additional drift after the local air temperature adjusts to the new CPU temperature (minutes). Discovering the fast initial response to p-state changes provided an opportunity to push the processor close to the thermal envelope; a p-state change can quickly correct any excursions beyond the limit. In well-cooled environments, CPU temperatures scale with DVFS p-states for a given workload activity as a result of a nearly lin-

ear relationship between power and temperature. Identifying the strong relationship between CPU temperature and DVFS states demonstrated the opportunity for thermal management via DVFS. We show the relationship between ambient and CPU temperatures, including drifting ambient conditions and harsh under-cooled environments. Our experiments indicate that a thermal control system would require information about the current thermal conditions, such as ambient temperature, in addition to power consumption. Simply predicting temperature from workload activity or measured power will not furnish sufficient information for effective thermal management.

Predictive Models We developed predictive models for estimating power for each p-state based on measured power for the present p-state. We developed two types of power models. The first uses hardware monitoring counters, also known as event counters and performance counters, to capture workload activity. Linear regression from offline training data provides a simple model with constant coefficients and counter values, useful for situations with a small number of event counters dedicated to power management. A second model also applies linear regression to offline training data to predict power for all p-states based on measured power directly, without event counters. This model is suited for scenarios in which event counters are not available for power monitoring, such as when the counter values are programmed for other event types. Both models are specific to the Pentium M 755 (90nm process, 2 GHz maximum frequency) processor. To our knowledge, we are the first to develop run-time power models for the Pentium M processor, and

the first to estimate power for multiple p-states on any platform.

8.1.3 Prototype

We incorporated predictive models and observations from our characterization studies into a prototype PET implementation to manage a Pentium M system. We demonstrated that the PET approach continuously adapts to workload behavior as it strives to meet target operating points. We present experiments throughout prototype development to demonstrate the capabilities of the prototype and opportunities and pitfalls of run-time power and thermal management with live hardware.

In one scenario, PET maximizes performance within power and thermal budgets by adapting DVFS settings to dynamic workload behavior with the SPEC CPU2000 suite with the `reference` input set. In this experiment, the objective function specifies "maximum performance." The hard limits are 70 °C, 17 Watts and soft limits are 43 °C and 14 Watts. In this case, the prototype manager generally adheres to hard constraints throughout execution and occasionally overshoots soft limits with minor excursions. The prototype manager violated soft limits for power 0.5% of the time, with a mean of 1 Watt overshoot and violated temperature 0.5% of the time, with a mean of 1.1 °C over the limit (with a temperature resolution of 1 °C). In the second scenario, both the workload behavior and the goal modulate during execution. The objective function remains "maximum performance" and the hard limits remain fixed, and the soft limits vary according to user-supplied software signals. In this experiment, the manager violates soft limits more often as it handles a wider range of operation; the manager does not meet soft-limit performance

requirements 0.3% of the time and power soft limits for 12.5% of execution time. Maximizing performance within a power-thermal envelope is a prerequisite task for high-performance systems. We apply this scenario to illustrate the prototype in a typical setting, although maximum performance is not the only goal.

We also demonstrate how the prototype PET manager gracefully handles a sudden drop in power limits, and how the PET prototype fulfills specified performance goals while conserving power and energy. Tradeoffs among performance, power, energy, and temperature will require prioritization and compromise. The PET approach provides a means to achieve those compromises. PET goal specifications are designed to guide operation to the desired outcome and continue to effectively manage the system as the goal, workload, and environment change over time.

The prototype goal-driven controller is able to interpret the goal specifications into DVFS p-states continually on a feasible time scale with negligible overhead, choosing reasonable settings in most cases. The level of indirection afforded by the goal specification and detailed information from performance event counters together provide a solution to specify true preferences at a high level, with the capability to enforce them at a lower level in the hardware stack. The prototype demonstrated the feasibility of the PET approach with a single-processor system. The prototyped approach holds promise for large-scale systems with a collection of PET managers that each control a local region and together manage a larger system. In a cluster-level PET manager, goal specifications would be determined by system administrators, either tuned by human operators or automatically updated

by operational utilities.

8.2 Future Directions

Power and thermal issues are driving forces in system design and operation. Power was a fundamental reason behind the shift from bipolar transistors to metal-oxide semiconductors (MOS) several years ago; as clock frequencies increased, bipolar technology's power handicap outweighed its advantage in smaller area. Technology continues to bring innovative devices, interconnect, materials, and packaging to the designer's toolbox. Leakage current, once considered to be a grave concern for future technology [9], has been tempered by transistor design and material changes; although transistors do consume static power from leakage current, the effect is manageable at the present and near-term future process generations.

Despite technological improvements, we expect performance-aware power and thermal management issues to persist. The cost in power, power density, heat, hotspots, reliability, and cooling provides a benefit in computation latency and throughput. Appetite for computing systems is not diminishing, and current trends indicate an ever-increasing amount of energy resources will be devoted to computing. Complex computing systems present a challenge for effective management: multiple processors, chipsets, networks, several levels of memory hierarchy from on-chip caches through disk drives, and multiple fans. Data centers house thousands of computers and supporting components. Each component and subsystem may exhibit unique behavior due to manufacturing variability and workload characteristics. Power and thermal management options throughout the system may have

overlapping jurisdictions due to virtualization and shared power supplies and cooling zones. Even independent management options can trigger complex interactions, as changing frequencies, capacities, and sleep modes alter resource availability and response times throughout the system.

Piecewise solutions do not provide sufficient management in isolation; multiple solutions operating independently do not necessarily yield the desired outcome. With increasing variability in component behavior due to manufacturing variation, conservative margins to ensure reasonable response for all parts would unnecessarily penalize faster or lower power individuals. The question becomes how to coherently manage multiple problems, and on multiple levels to make the best possible use of the work of electrons.

A collection of PET managers could be a viable option for controlling individual chips with leakage control techniques and DVFS, tailoring memory capacity with sleep modes for SRAMs and spin-down modes for hard drives, and even managing air flow for fans and computer room air-conditioning units. Individual managers would share information with other managers to share resources and meet common goals, each each operating on a timescale suitable for its own jurisdiction. Large-scale systems such as supercomputers and data centers could be partitioned into zones of autonomous PET management, tailoring goals within each zone to different purposes to allow maximum throughput for high-priority workloads in some partitions and improved energy efficiency for others.

In our research, we have taken steps toward spending power wisely, by providing a framework to achieve the desired outcome without conservative safety mar-

gins or wasting power with ineffective solutions. We have identified the need for—and the challenges involved with—coherent, run-time performance-aware power and thermal management. We focused on microprocessor power with one level of PET hierarchy for a detailed view of the problem and potential solutions. The multi-dimensional nature of the PET goals could accommodate future needs for additional management objectives, such as reliability and security, as well.

Appendices

Appendix A

Power and Performance Characterization Data

We characterized application response to two power management techniques, dynamic voltage and frequency scaling (DVFS) and clock throttling, applied individually and in combination on a Pentium M system with a set of microbenchmarks designed to capture core and memory subsystem activity. We analyzed microbenchmark power and performance for three dataset sizes corresponding to core-bound, intermediate, and memory-bound programs. We found that for techniques applied individually, DVFS is more effective than clock throttling at reducing power while preserving performance. We also applied both techniques together in each of the 64 combinations of 8 frequencies and 8 clock throttling levels. We demonstrated that a performance or power target can be achieved with multiple combinations of DVFS and clock throttling settings, however, DVFS provides superior power-performance efficiency, especially with memory-bound applications.

A.1 Dynamic Voltage and Frequency Scaling

Figures A.1 through A.6 illustrate the effect of DVFS on power consumption and performance for the L1-resident 4 KB footprint, the L2-resident 128 KB footprint, and the main-memory (non-cache resident) 4 MB data footprints of the microbenchmark suite. Performance is charted in terms of inverse program execution

time, normalized to the maximum performance (minimum execution time) for each benchmark-footprint combination for a fair comparison of the microbenchmarks, which vary in program length. The core-bound and intermediate footprints show very little application-specific behavior variation throughout the DVFS spectrum and normalized performance is linear with frequency. The memory-bound footprint, however, is greatly influenced by frequency settings. As the core frequency is reduced, the relative speed of the core with respect to memory slows and the core waits fewer cycles for data from memory, effectively reducing the performance penalty of lower frequencies. The extent of memory influence is application-specific, with the `FMA` microbenchmark incurring the largest performance degradation.

For these microbenchmarks, power consumption ranges from 14-17 Watts at the 2 GHz highest-frequency setting to about 3 Watts at 600 MHz, the lowest frequency. Benchmarks with L1- and L2-resident footprints display a quadratic relationship between the p-state setting and power, with more variation among benchmarks at high frequencies. The memory-bound 4 MB footprint also reflects the quadratic relationship, with a wider range of application behavior throughout the frequency spectrum than the core-bound and intermediate footprints, with increasing spread in power consumption at higher frequencies. In all footprints, the `MLOAD-RAND` test exhibits lower power consumption than the other microbenchmarks, in part because its random behavior does not benefit from pre-fetching and consequently spends more time stalling for memory accesses. In a system such as Pentium M with aggressive clock-gating for idle components, stall time translates to lower power consumption.

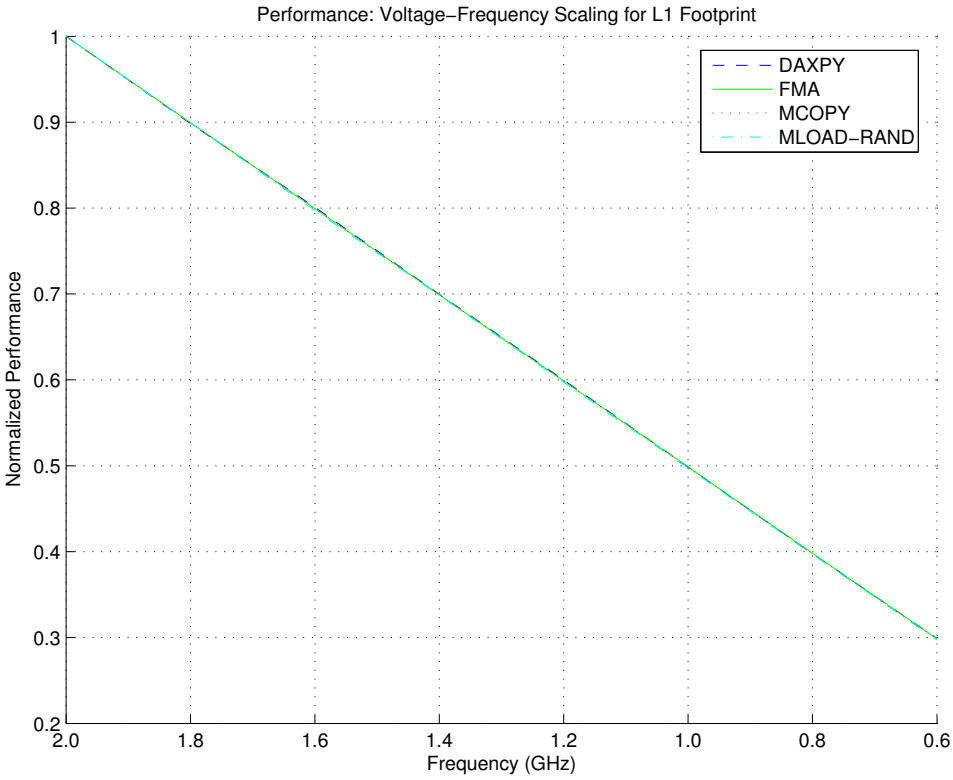


Figure A.1: Performance response to DVFS for 4 microbenchmarks with L1 data footprint.

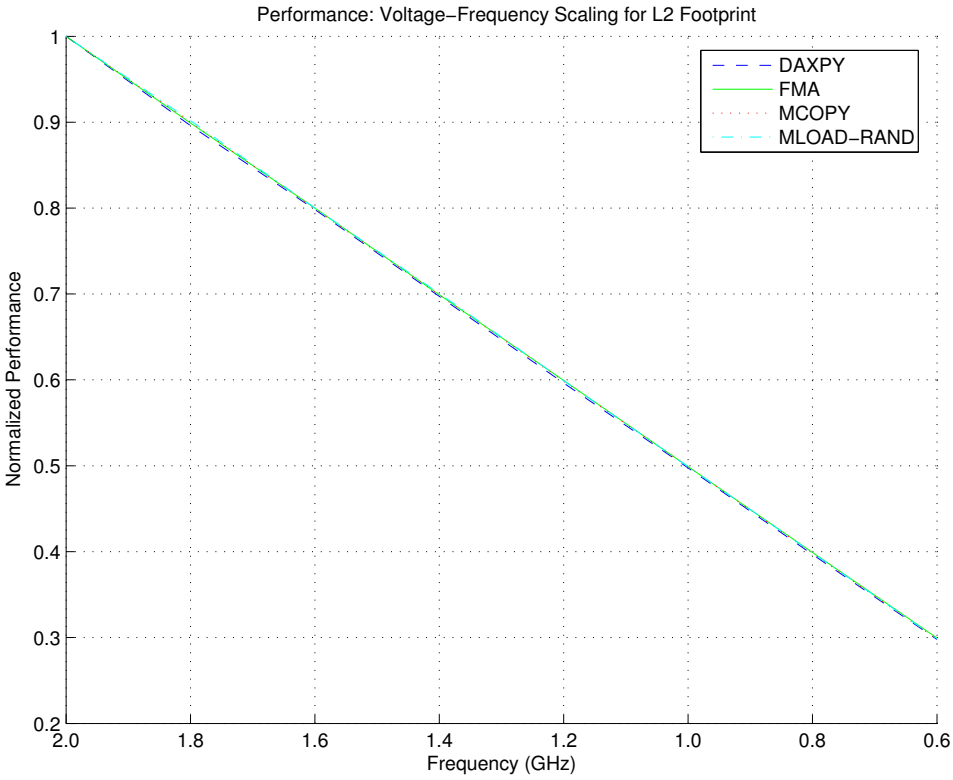


Figure A.2: Performance response to DVFS for 4 microbenchmarks with L2 data footprint.

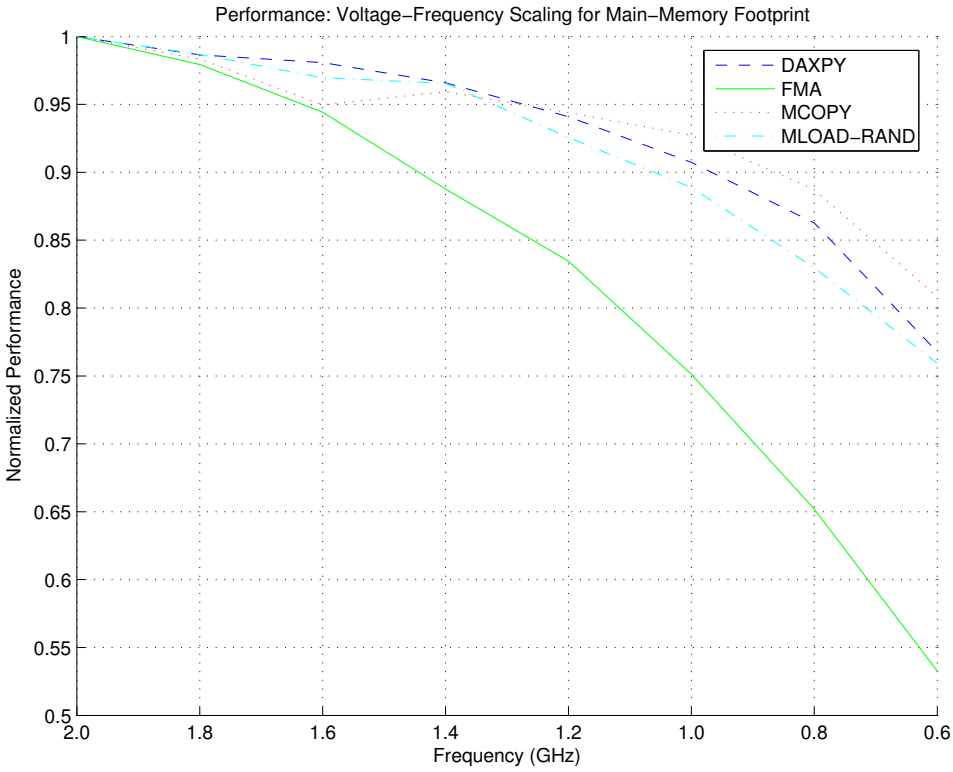


Figure A.3: Performance response to DVFS for 4 microbenchmarks with main-memory data footprint.

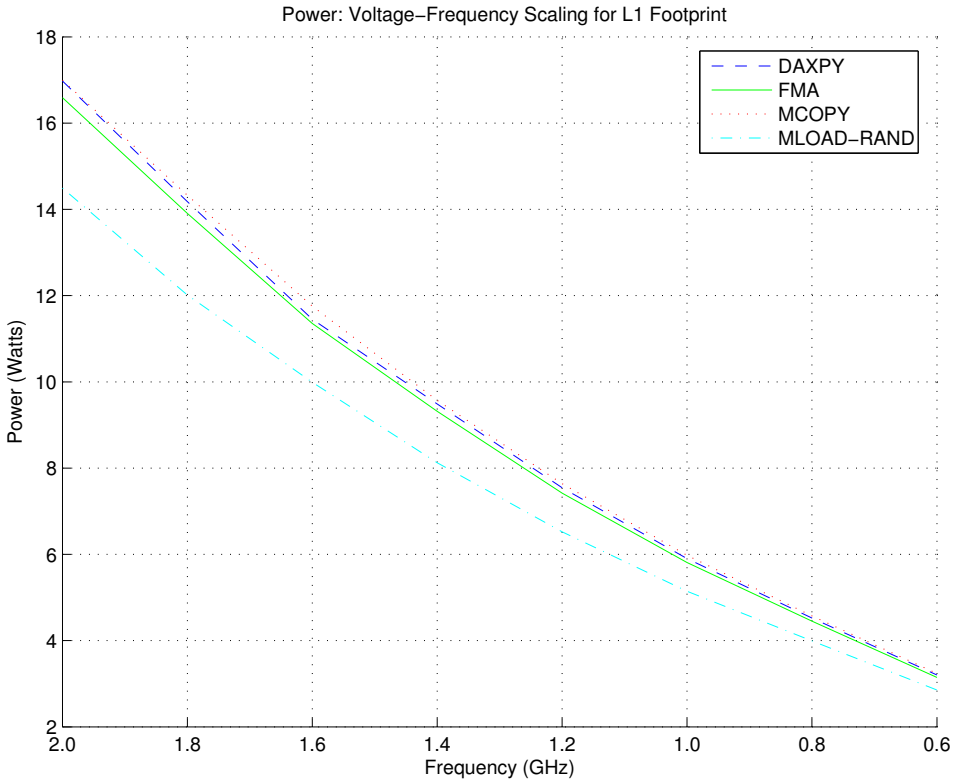


Figure A.4: Power for each DVFS p-state for 4 microbenchmarks with L1 data footprint.

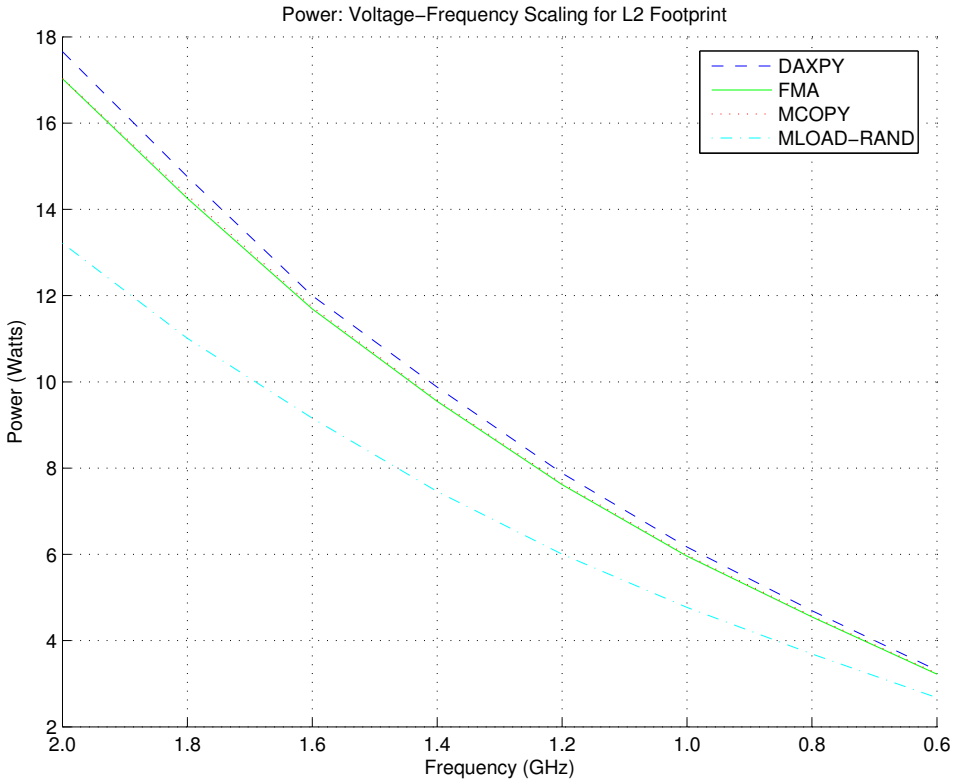


Figure A.5: Power for each DVFS p-state for 4 microbenchmarks with L2 data footprint.

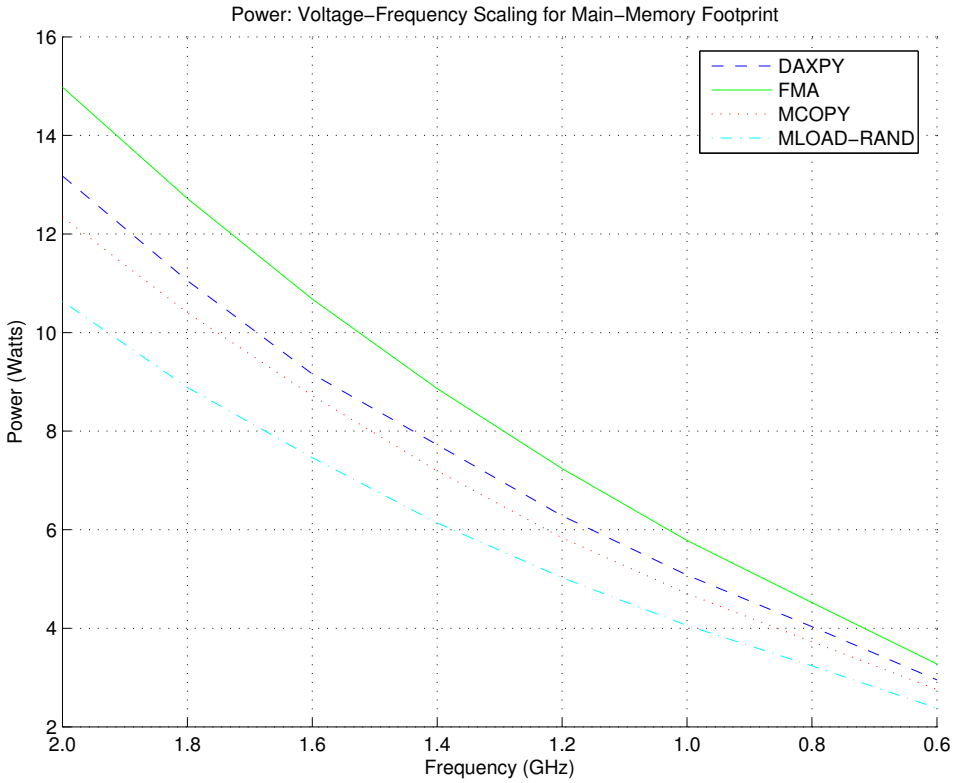


Figure A.6: Power response to DVFS for 4 microbenchmarks with main-memory data footprint.

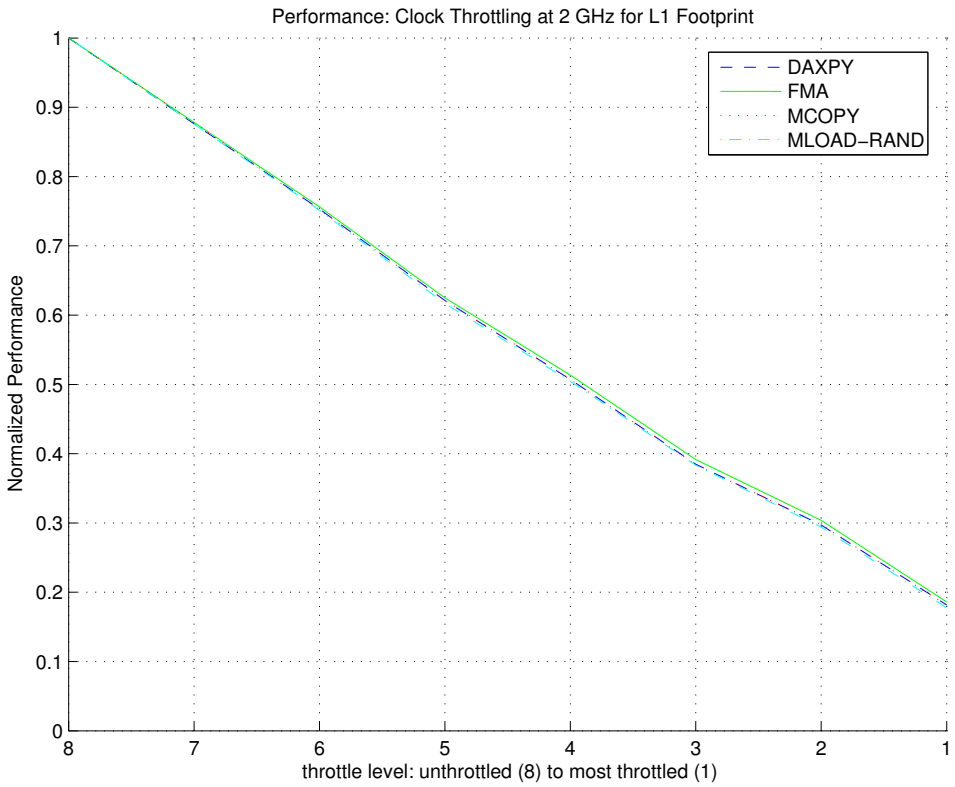


Figure A.7: Performance response to clock throttling for 4 microbenchmarks with L1 data footprint.

A.2 Clock Throttling

Figures A.7 through A.12 show the effect of clock throttling on power consumption and performance for microbenchmarks executing in each memory footprint at 2 GHz. Clock throttling with large $run + hold$ windows such as this system windows, does not alter the relative speed of memory with respect to the core frequency. The performance response to clock throttling is approximately linear with throttling extent for all data footprint sizes, roughly tracking the 12.5% steps in throttle val-

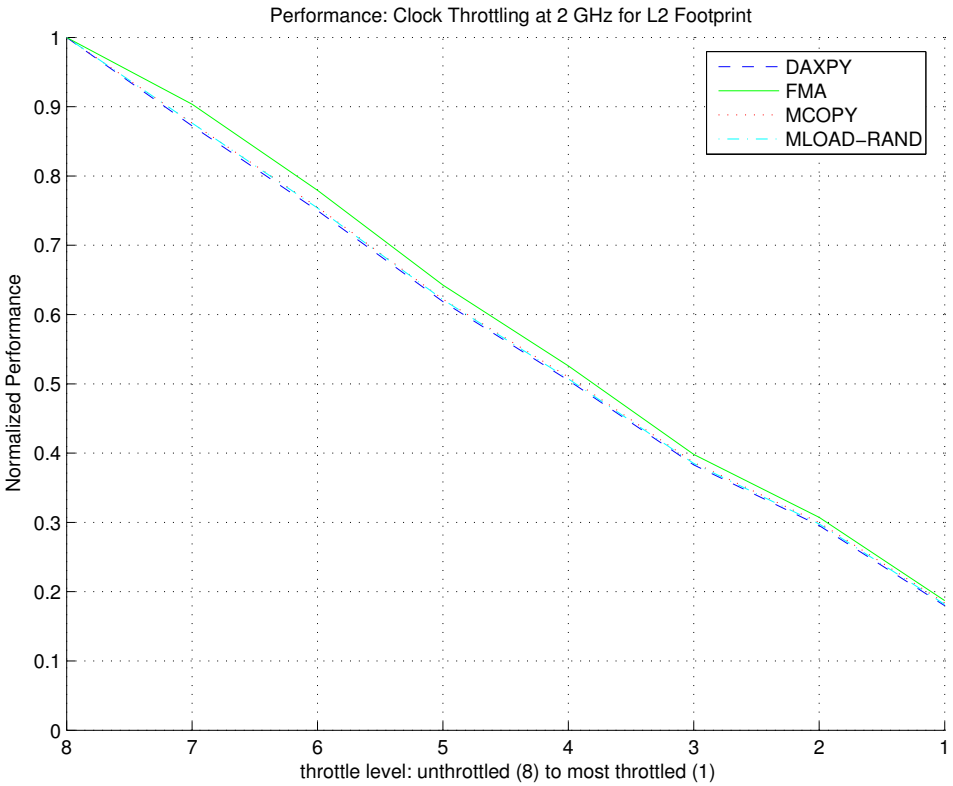


Figure A.8: Performance response to clock throttling for 4 microbenchmarks with L2 data footprint.

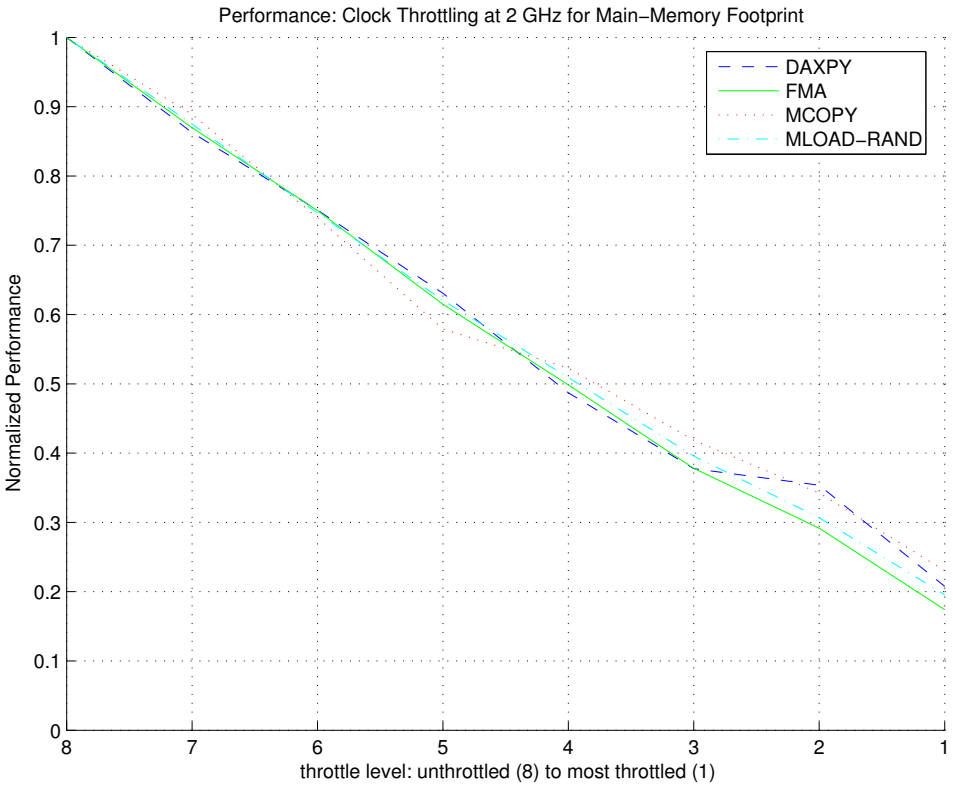


Figure A.9: Performance response to clock throttling for 4 microbenchmarks with main-memory data footprint.

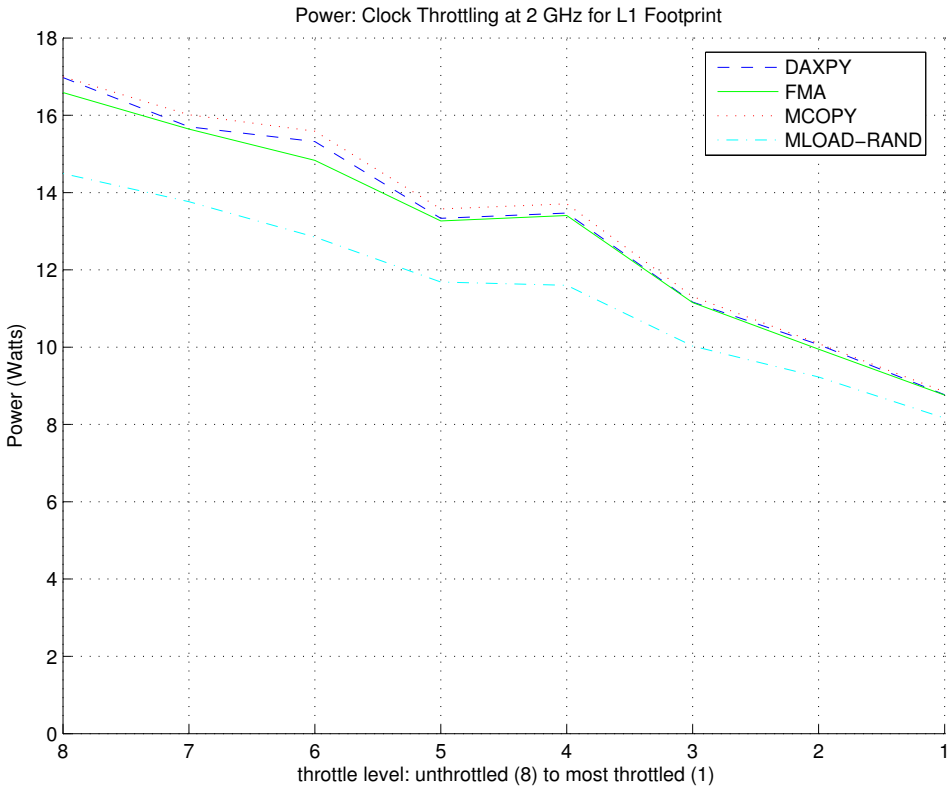


Figure A.10: Power response to clock throttling for 4 microbenchmarks with L1 data footprint.

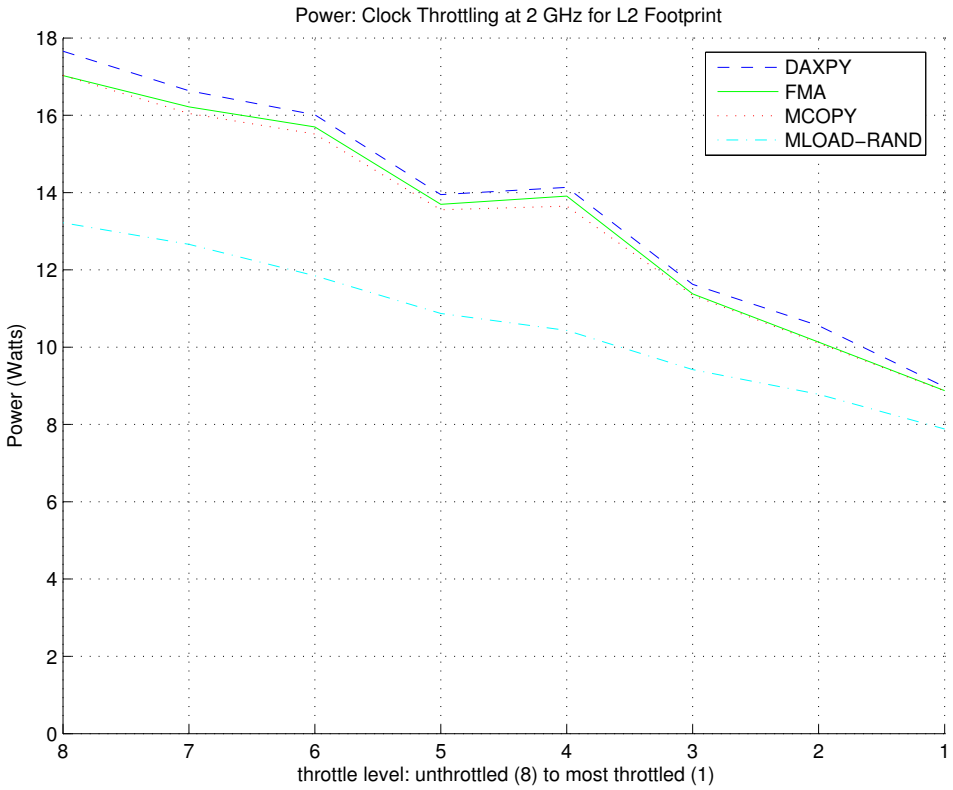


Figure A.11: Power response to clock throttling for 4 microbenchmarks with L2 data footprint.

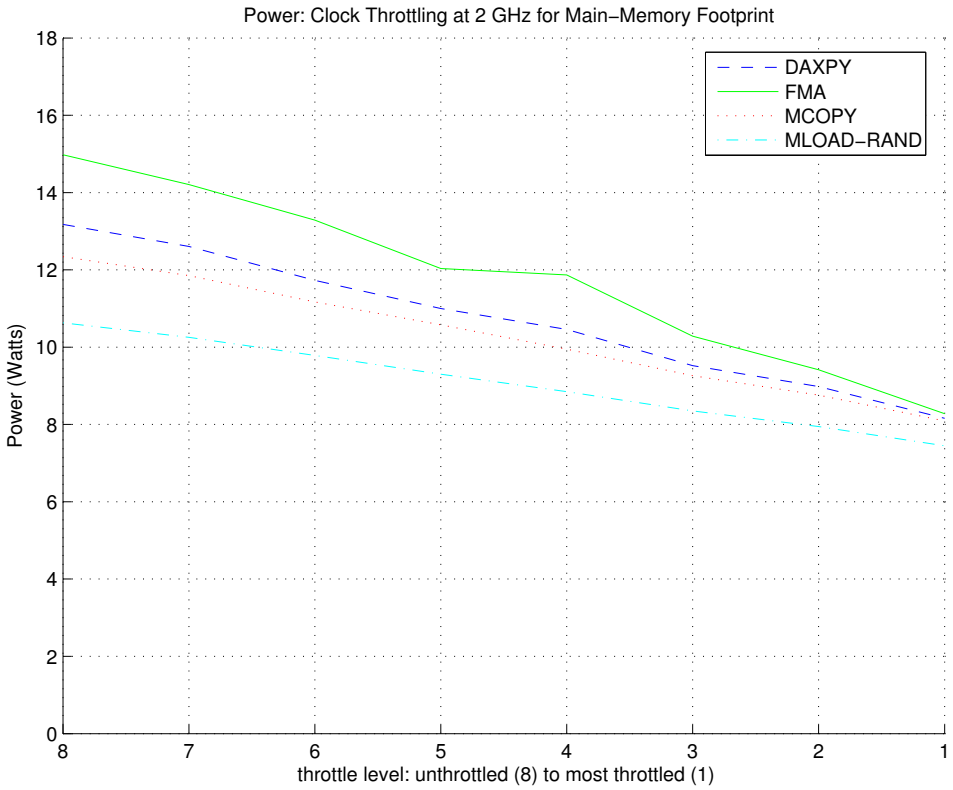


Figure A.12: Power response to clock throttling for 4 microbenchmarks with main-memory data footprint.

ues. Slight variations between benchmarks at larger footprint sizes are likely due to paging and other memory system effects.

Figures A.10 through A.12 indicate that mean power consumption for throttle level 4 is approximately the same as throttle level 5. An examination of the power and performance data suggested that the implementation of throttle level 4 on this system does not conform to the expected throttling behavior for a 50% duty cycle. Measured power at throttle level 4 exhibits a wide band of power, with a mean similar to throttle-level 5 but wider variation. Throttling level 5 is also somewhat more throttled than expected for a 5/8 duty cycle.

Aside from the mid-range throttling abnormalities, power consumption reflects an approximately linear relationship with clock throttling level, unlike the non-linear DVFS power trend. The lowest clock throttling setting does not control power to the same extent as the lowest frequency/voltage pair of DVFS settings. Clock throttling power consumption ranges from about 7.5 Watts at the maximum throttling to about 18 Watts unthrottled.

Clock-throttling power trends are similar to DVFS power trends with respect to variation: `MLOAD-RAND` consumes less power than the other microbenchmarks across footprint size and throttling settings and benchmarks with larger footprints and under less-throttled conditions exhibit larger power variation.

Appendix B

Pentium M Power and Performance Trace Data

During the course of our investigations, we monitored execution of each benchmark in the SPEC CPU2000, at each DVFS p-state. Figures B.1 through B.26 present power and performance data in alphabetical order by benchmark name, The figures illustrate several characteristics of power and performance:

- Few programs exhibit distinct power phases at the 10ms-per-sample timescale.
- For this system's specified frequency and voltage pairs, power consumption does not follow the popular "cubic rule" approximation, which would expect that doubling the frequency would increase power by 2^3, or 8x. Instead, doubling the frequency from 1000 MHz to 2000 MHz increases the frequency by a factor of approximately 3x. The cubic approximation depends in part on different voltage scaling expectations than the supply voltages supported for this system.
- For most programs, power and IPC deviate from mean values throughout execution.
- The extent of DVFS influence on execution time varies by benchmark; some are frequency sensitive (`apsi`), others are less sensitive (`swim`). Memory-bound benchmarks exhibit higher IPC values at lower-frequency p-states while

compute-bound benchmarks maintain more constant IPC values throughout DVFS p-states.

The figures plot power measurements for each p-state and performance measurements for only three p-states: the maximum 2000 MHz, an intermediate 1200 Mz, and the minimum 600 MHz (to reduce overlapping data on the instructions per cycle graphs).

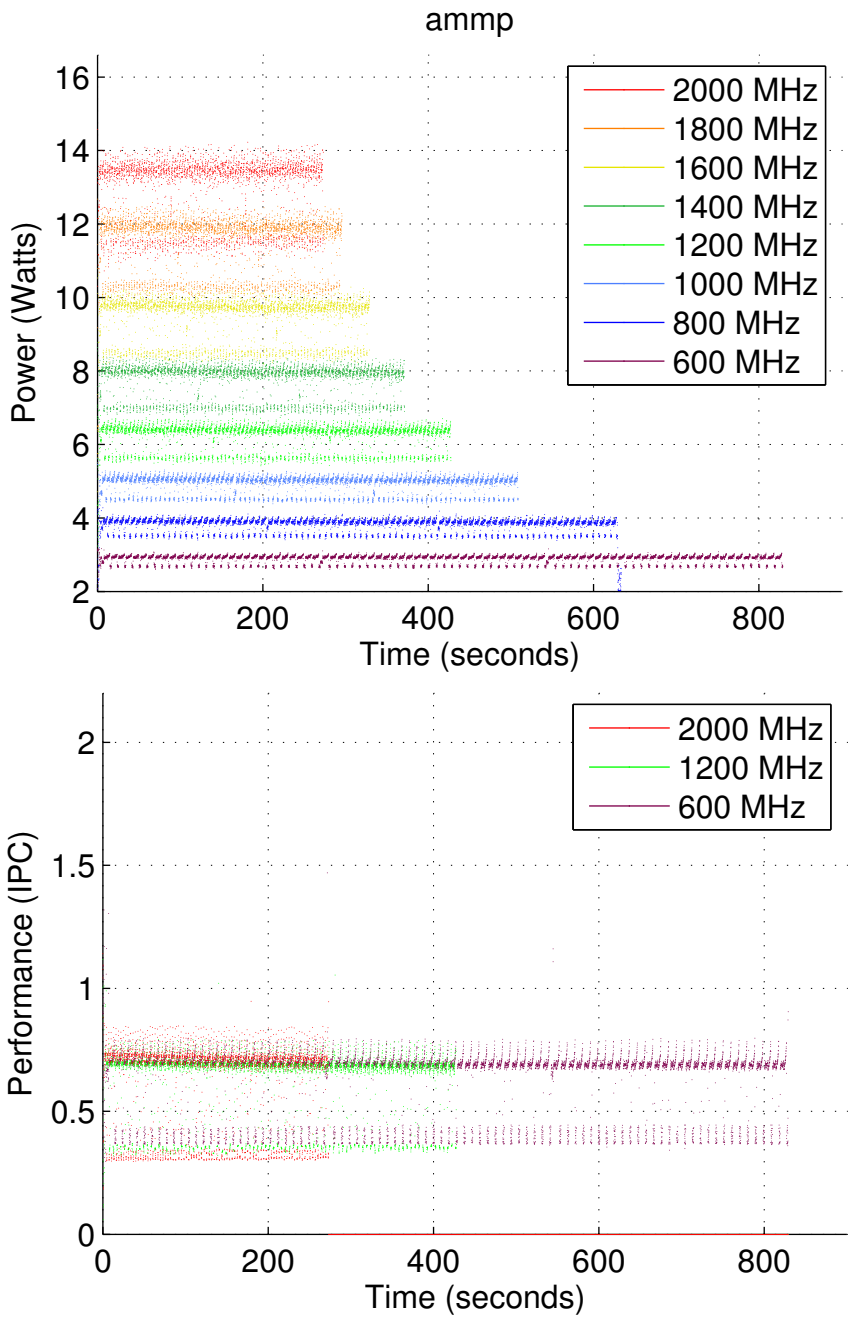


Figure B.1: Power and IPC for ammp

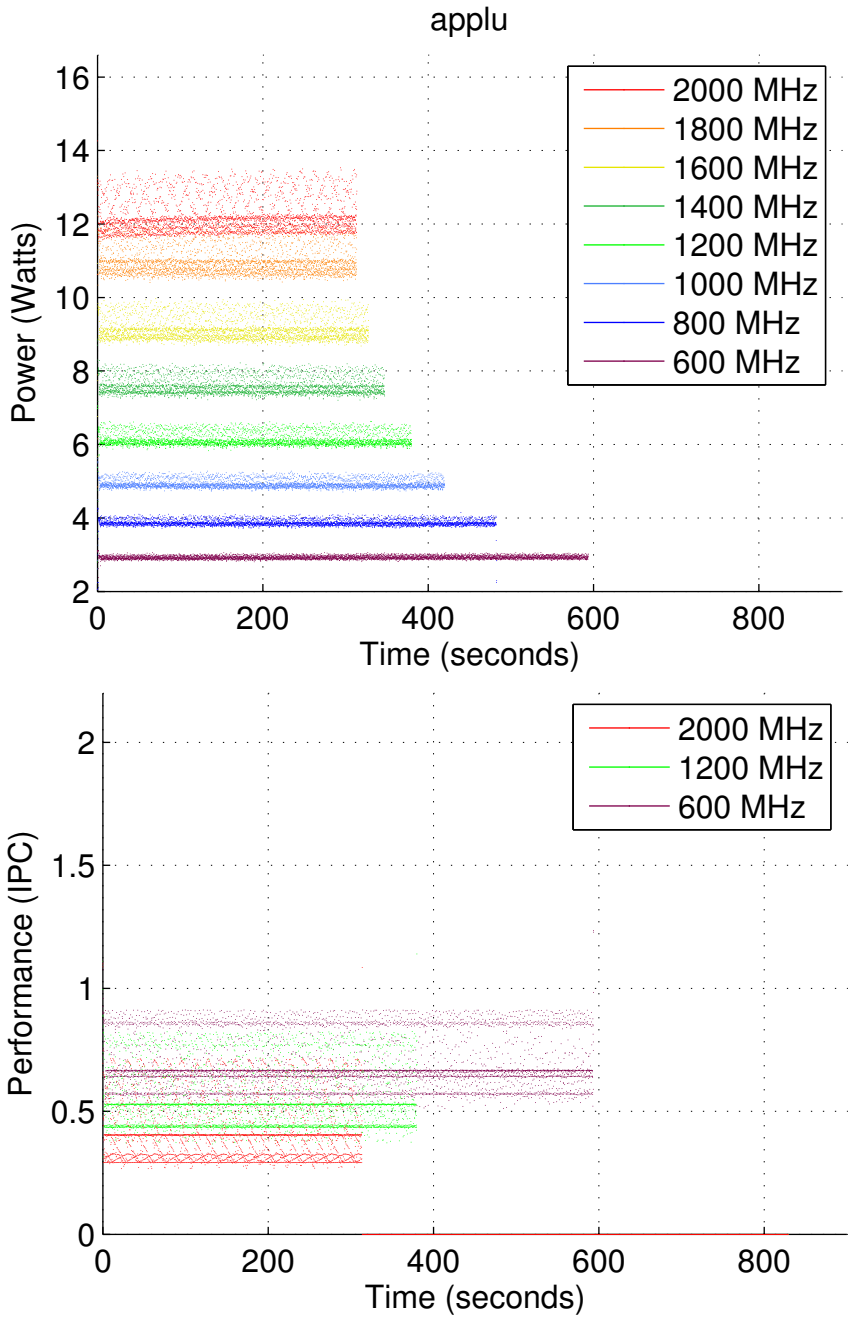


Figure B.2: Power and IPC for applu

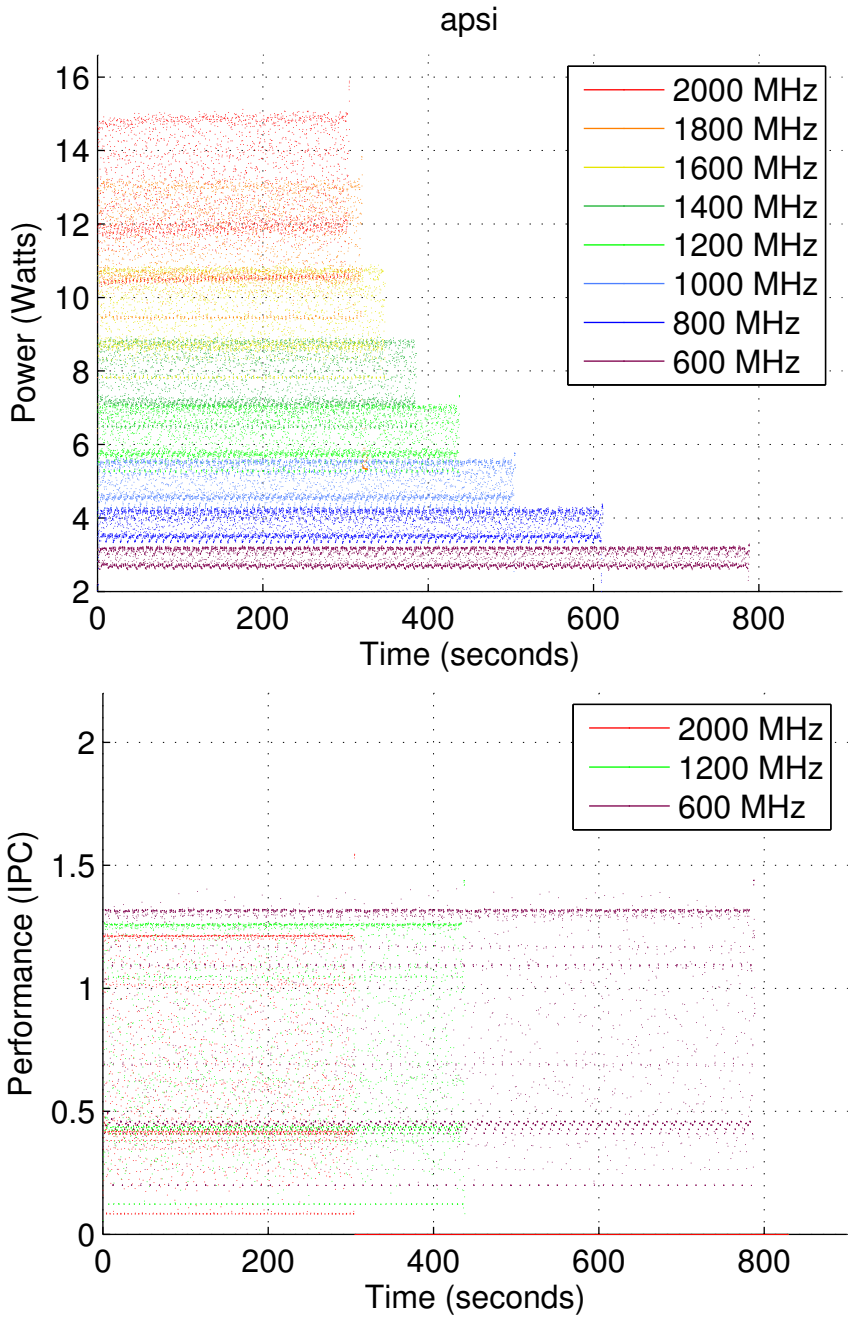


Figure B.3: Power and IPC for apsi

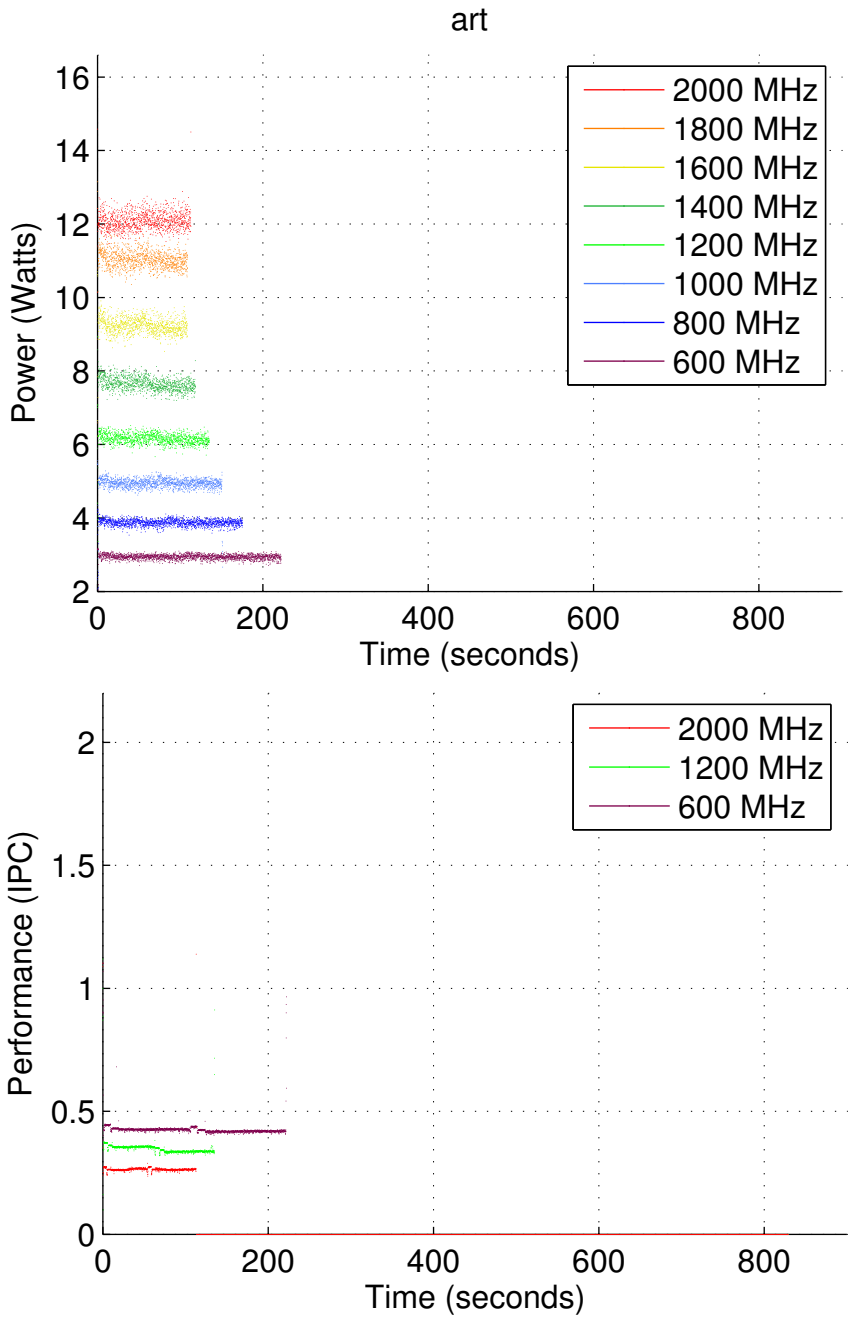


Figure B.4: Power and IPC for art

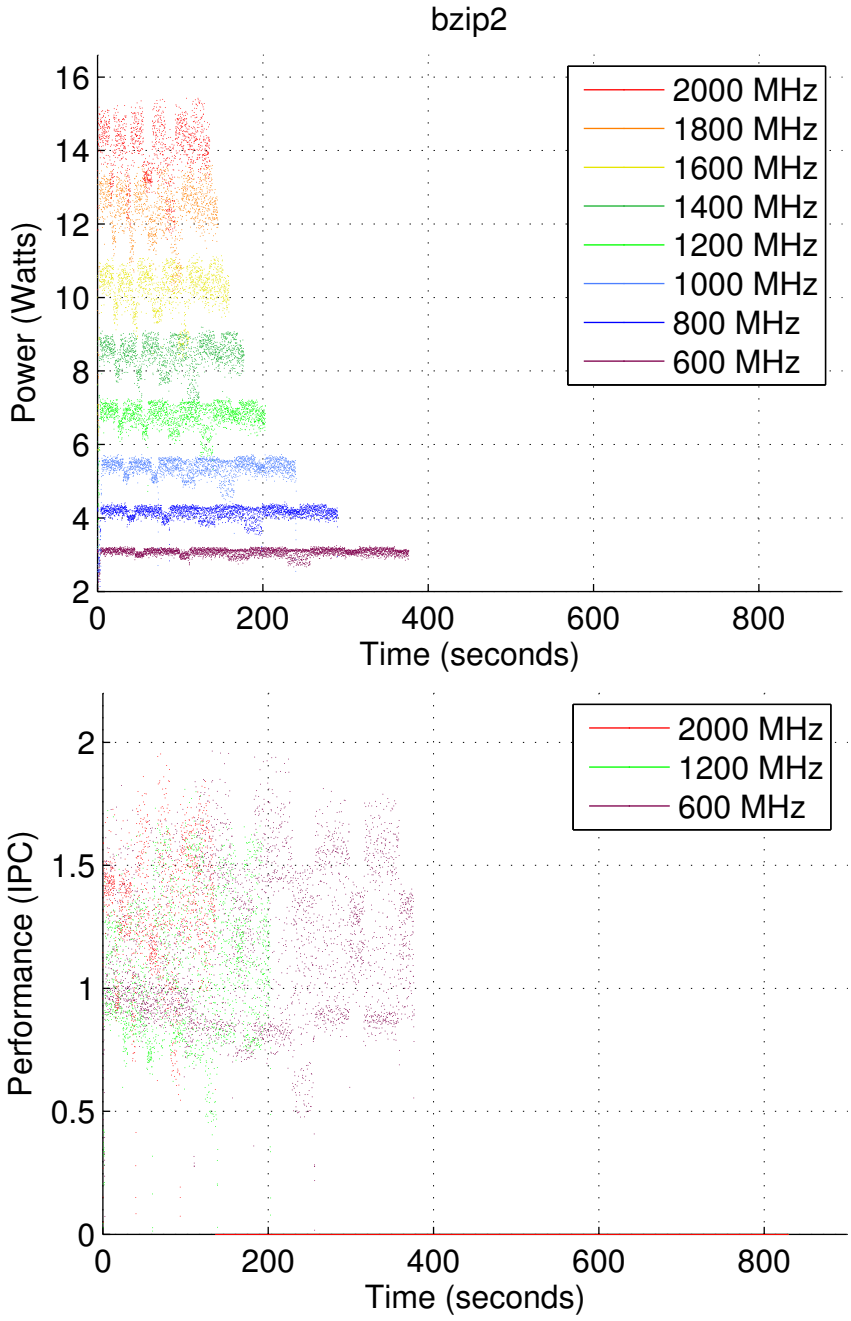


Figure B.5: Power and IPC for bzip2

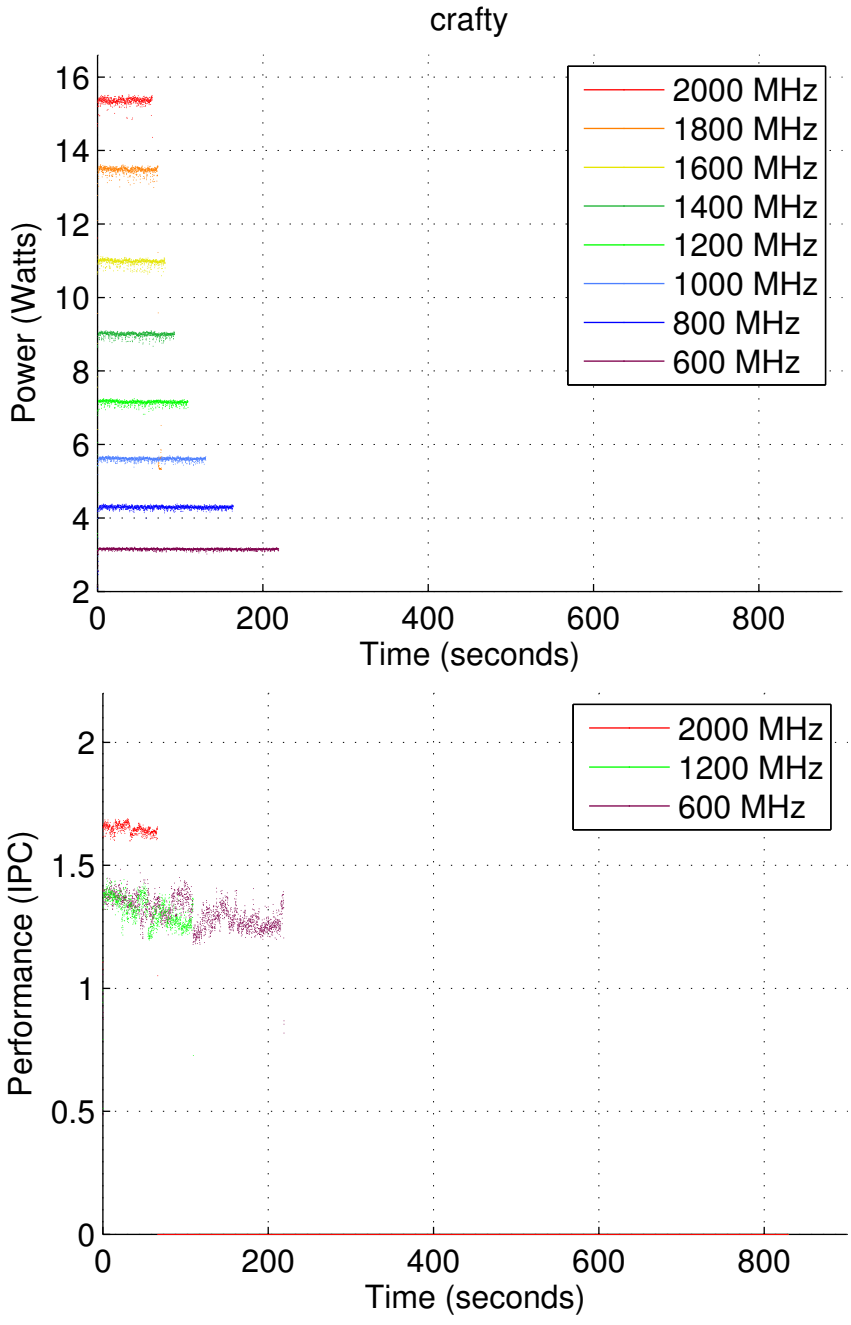


Figure B.6: Power and IPC for crafty

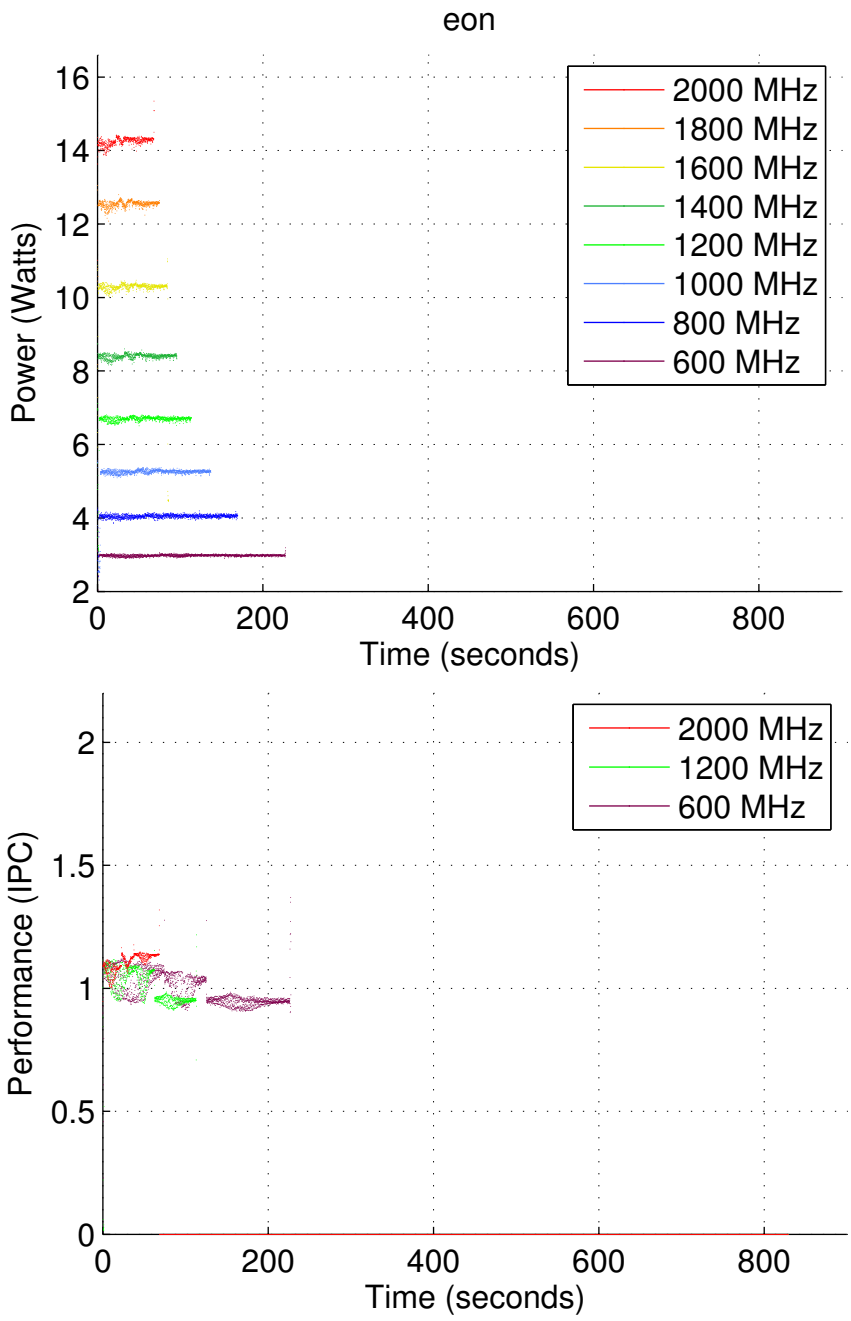


Figure B.7: Power and IPC for eon

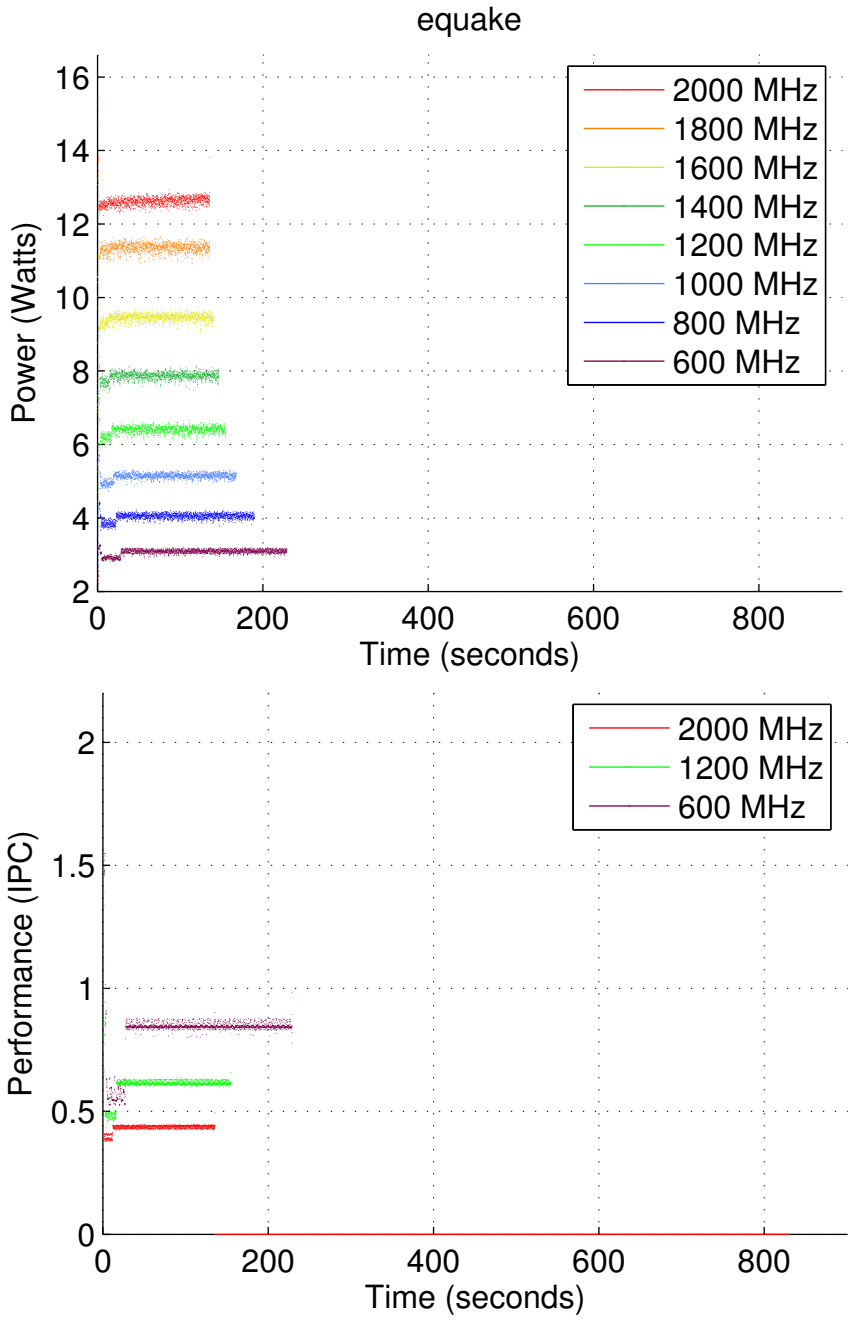


Figure B.8: Power and IPC for equake

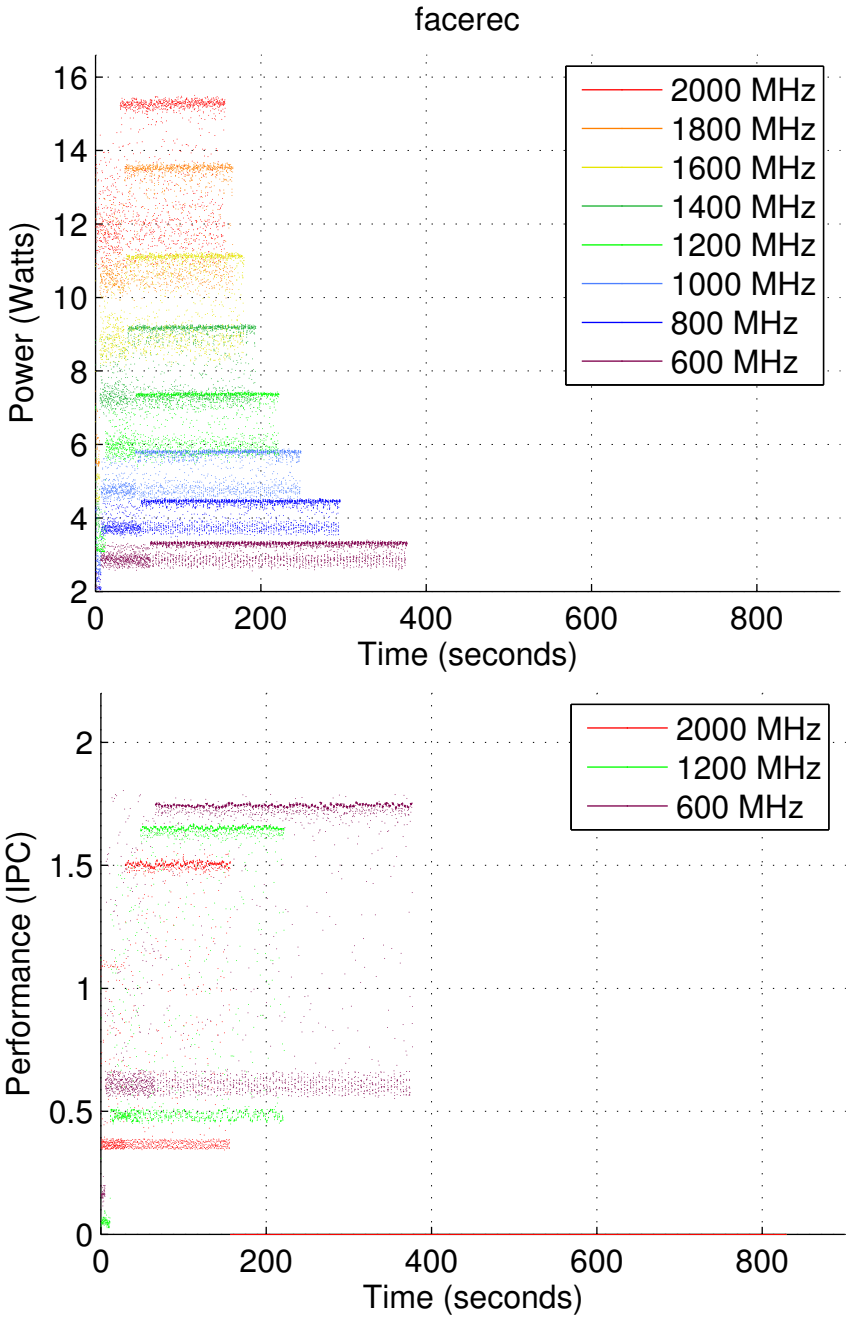


Figure B.9: Power and IPC for facerec

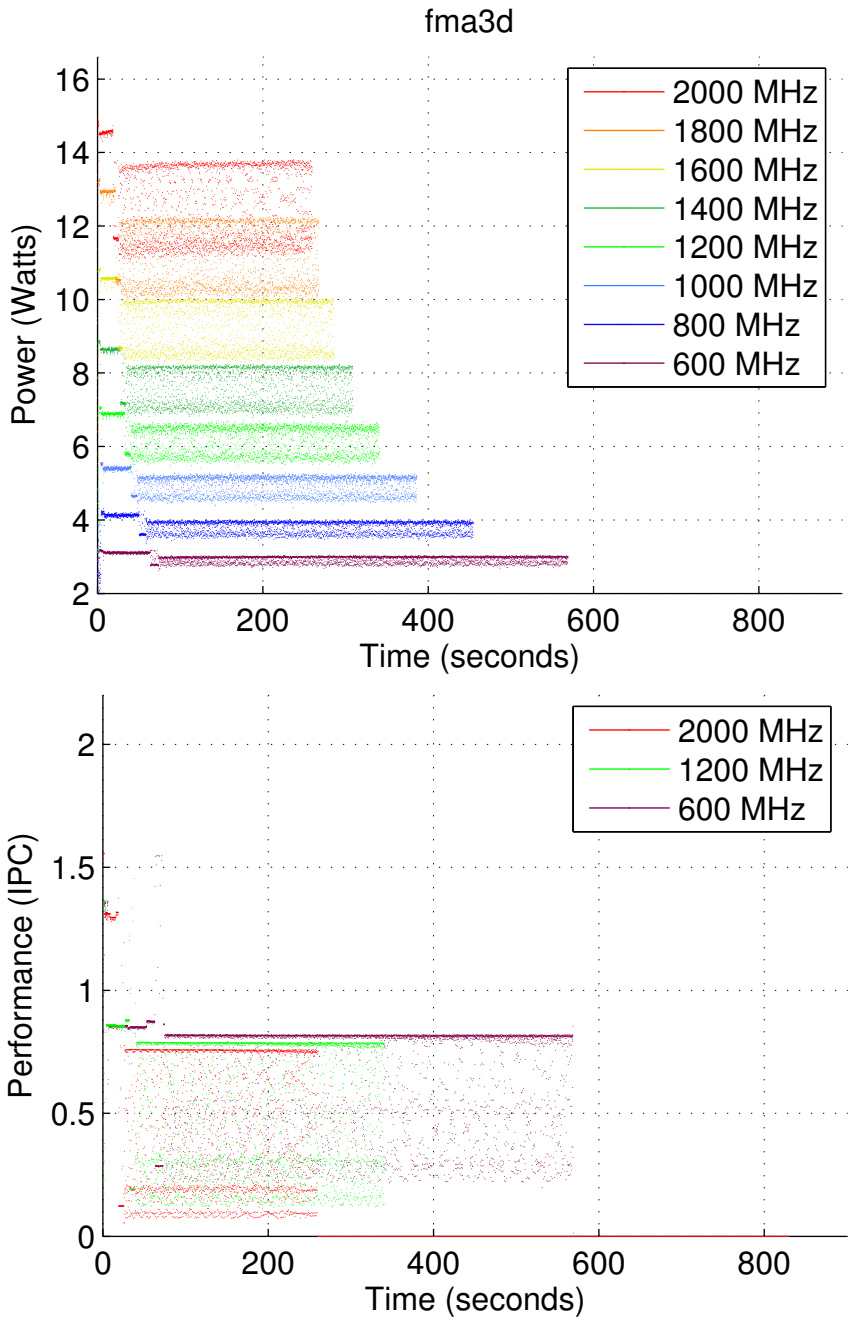


Figure B.10: Power and IPC for fma3d

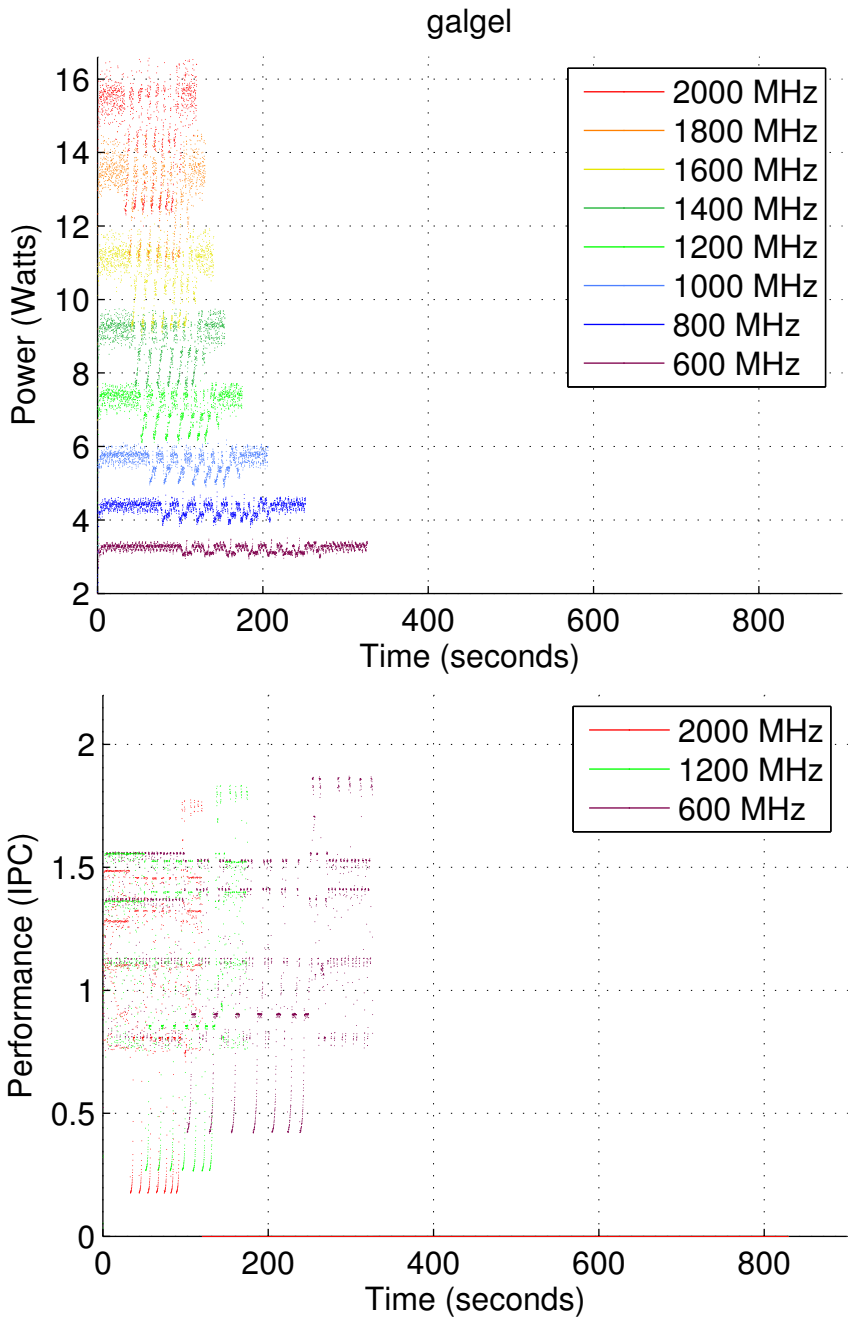


Figure B.11: Power and IPC for galgel

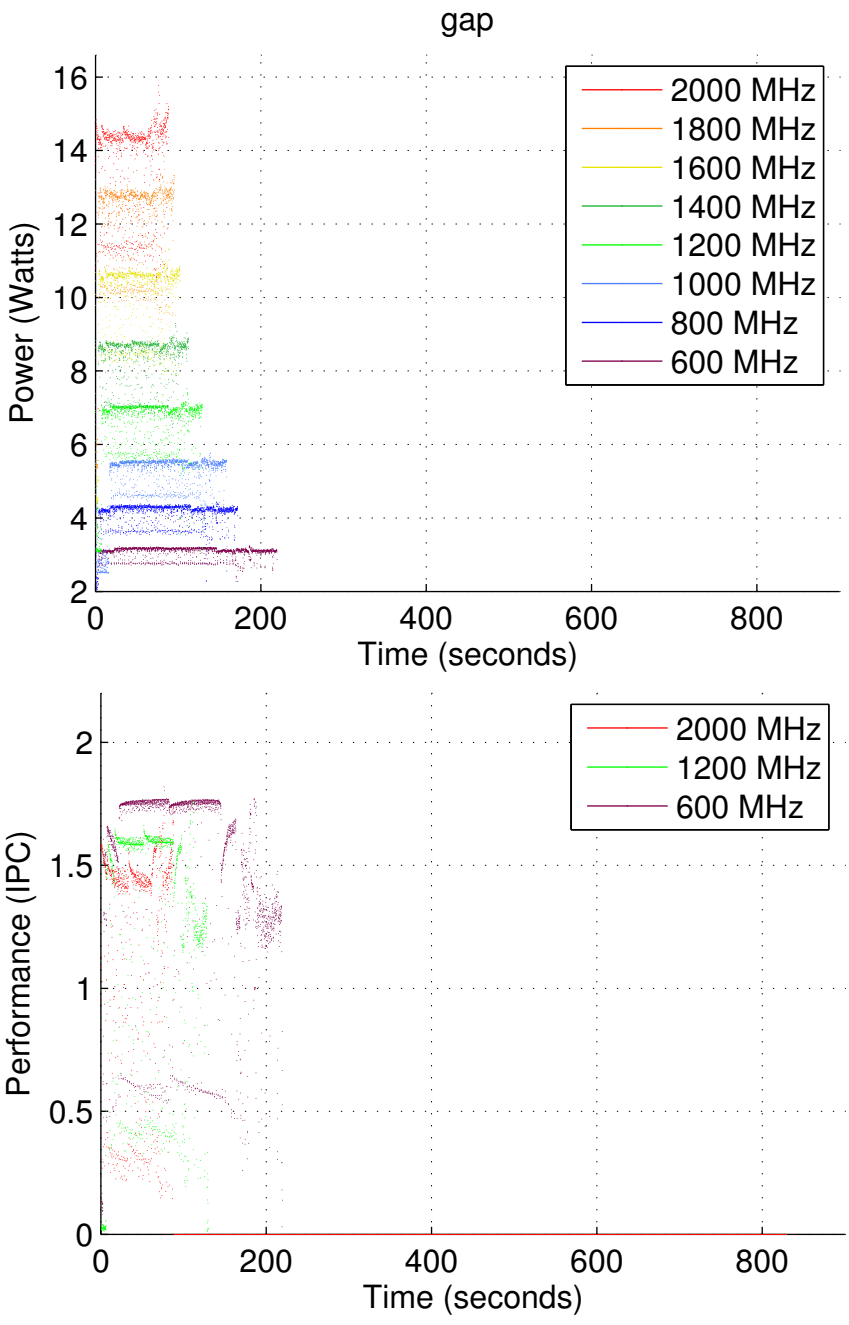


Figure B.12: Power and IPC for gap

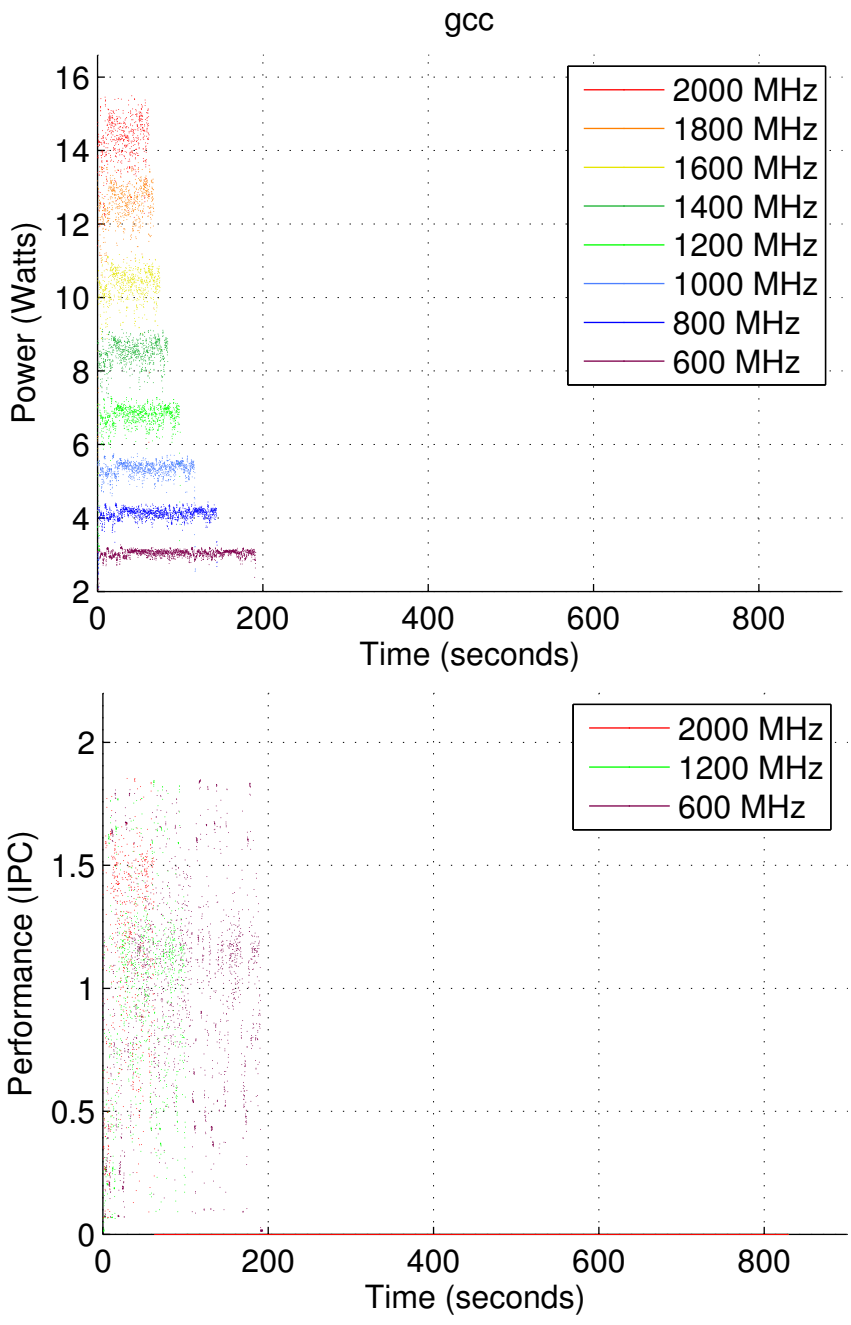


Figure B.13: Power and IPC for gcc

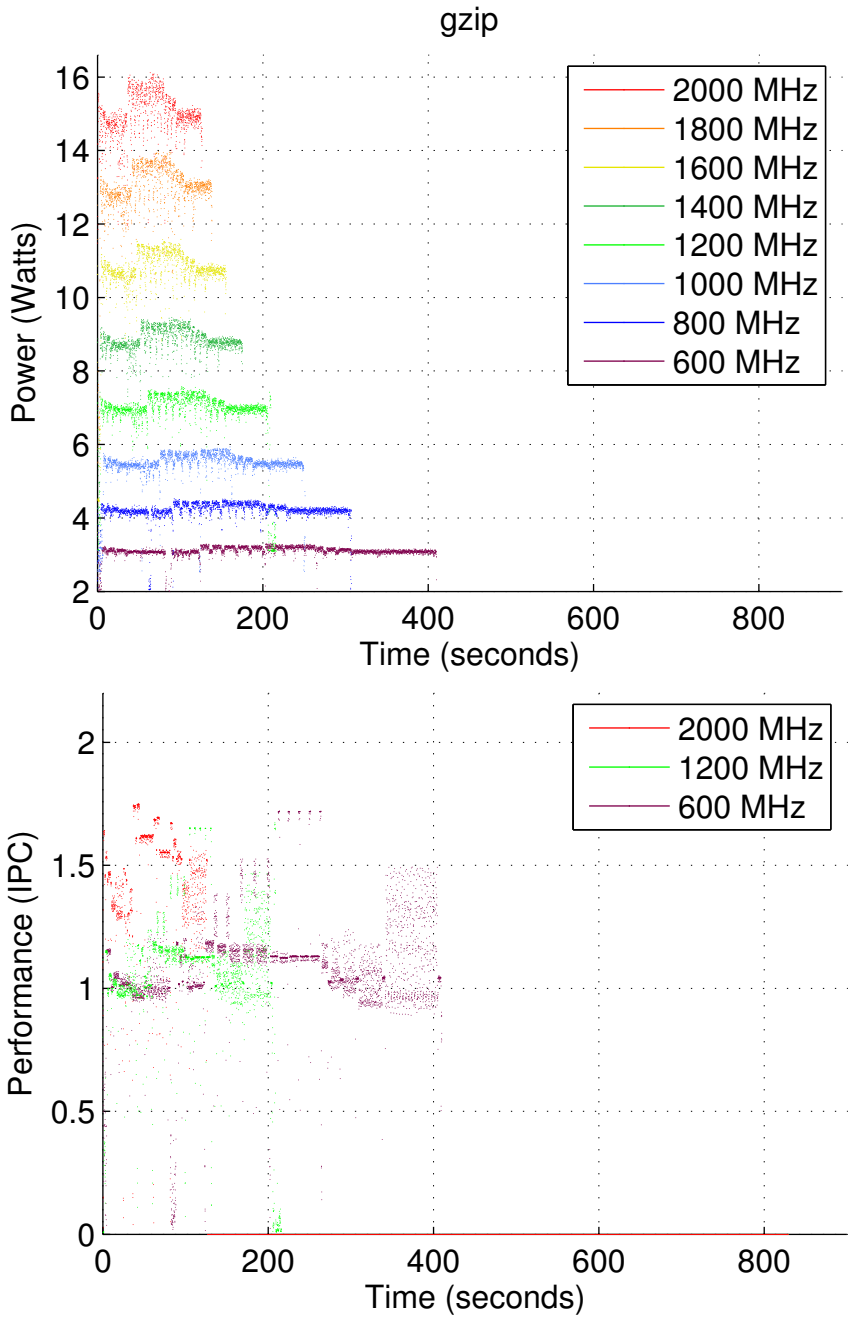


Figure B.14: Power and IPC for gzip

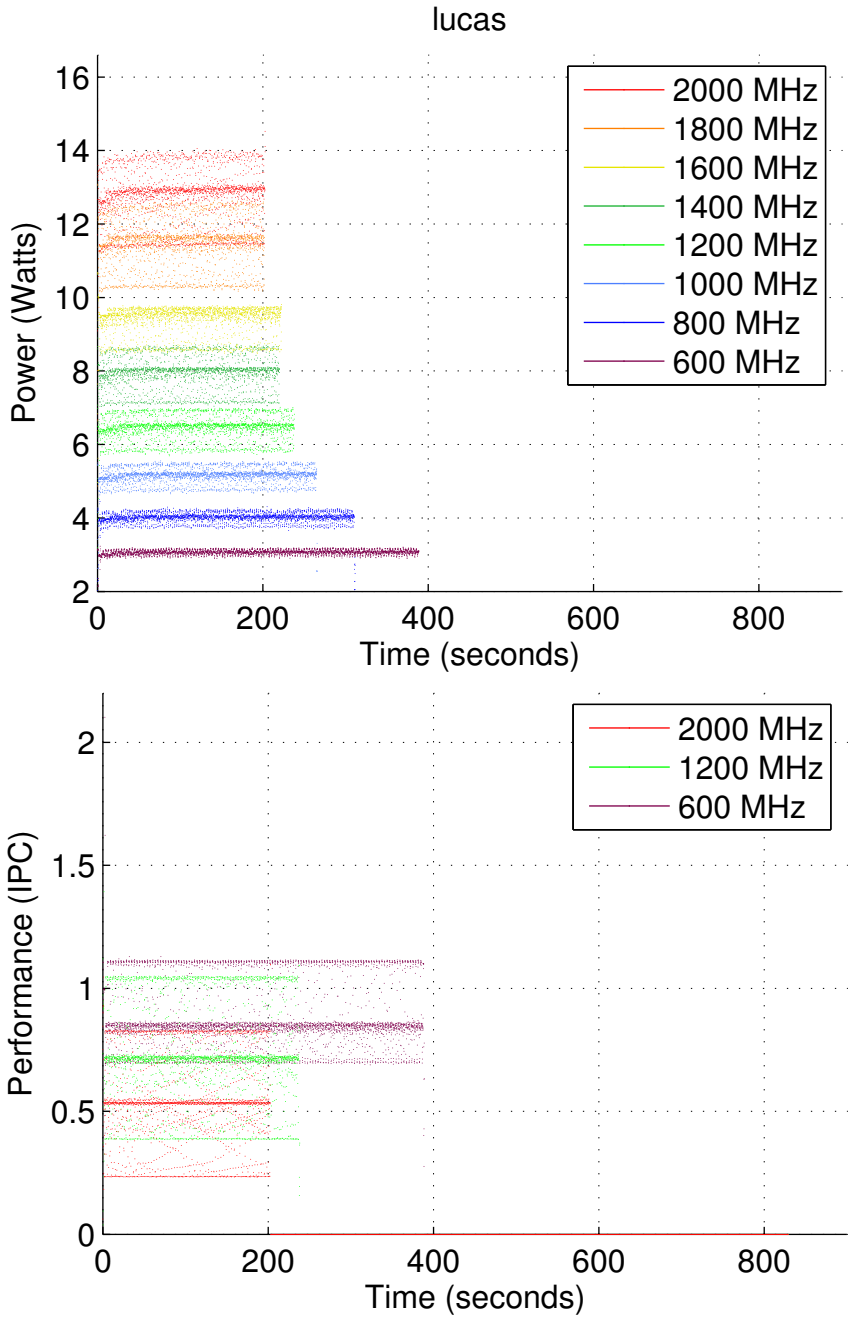


Figure B.15: Power and IPC for lucas

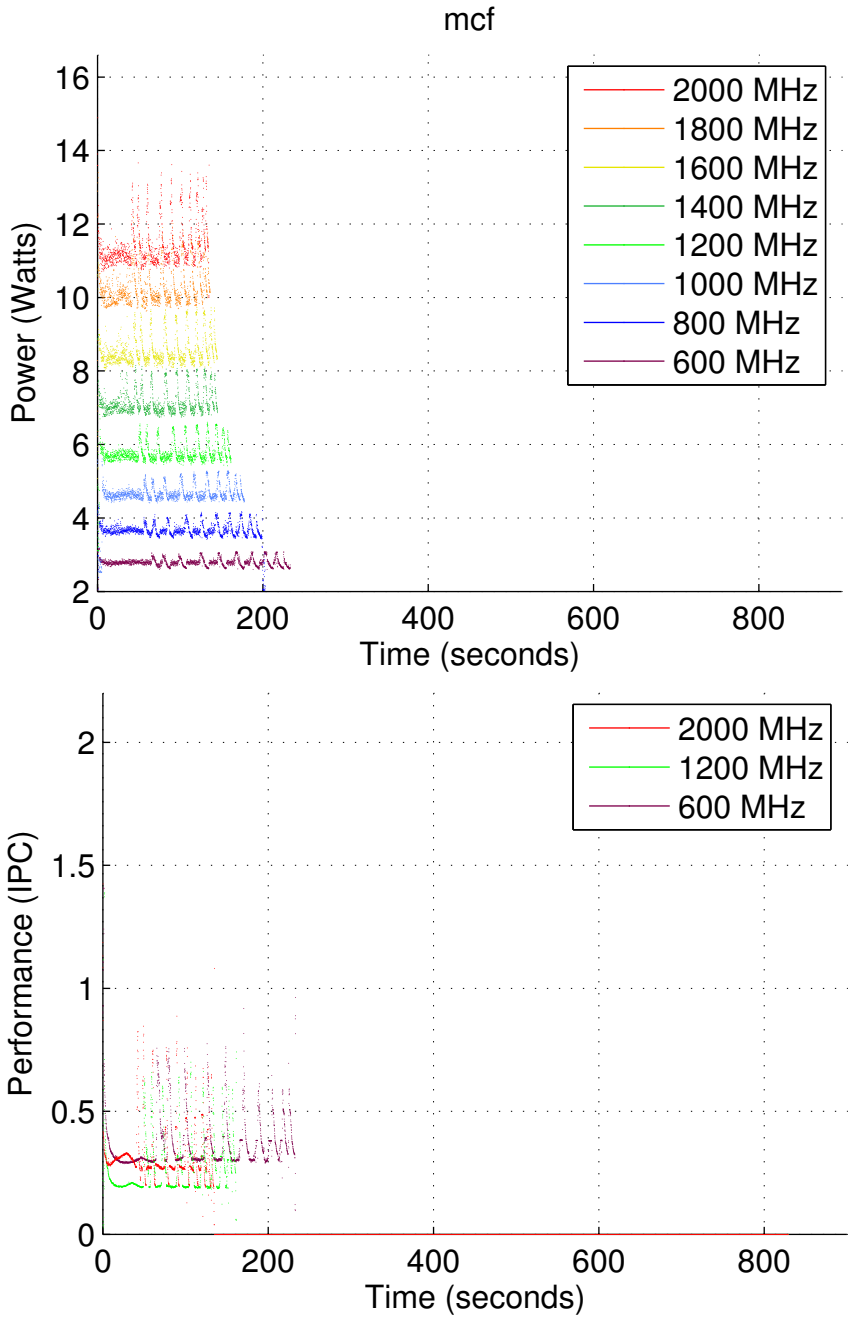


Figure B.16: Power and IPC for mcf

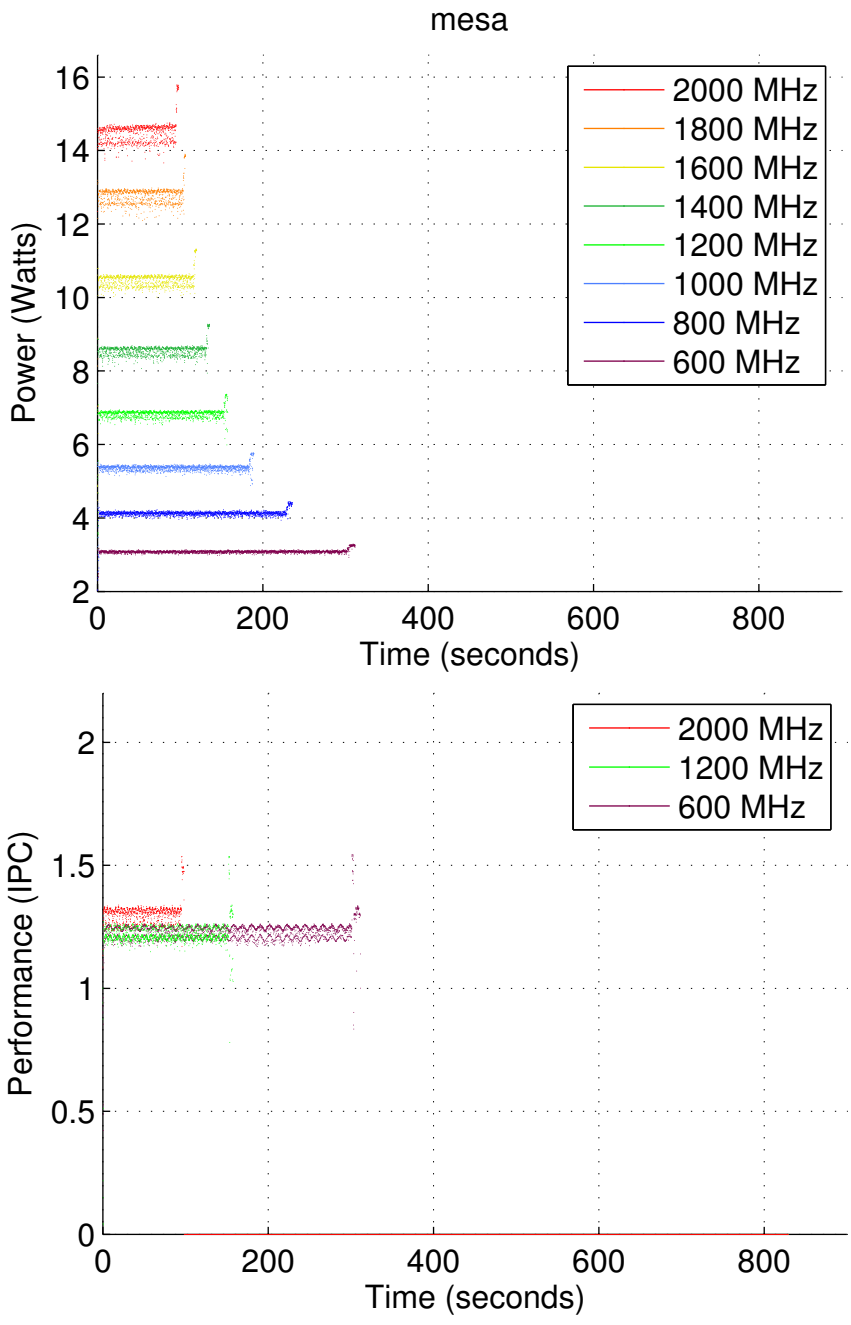


Figure B.17: Power and IPC for mesa

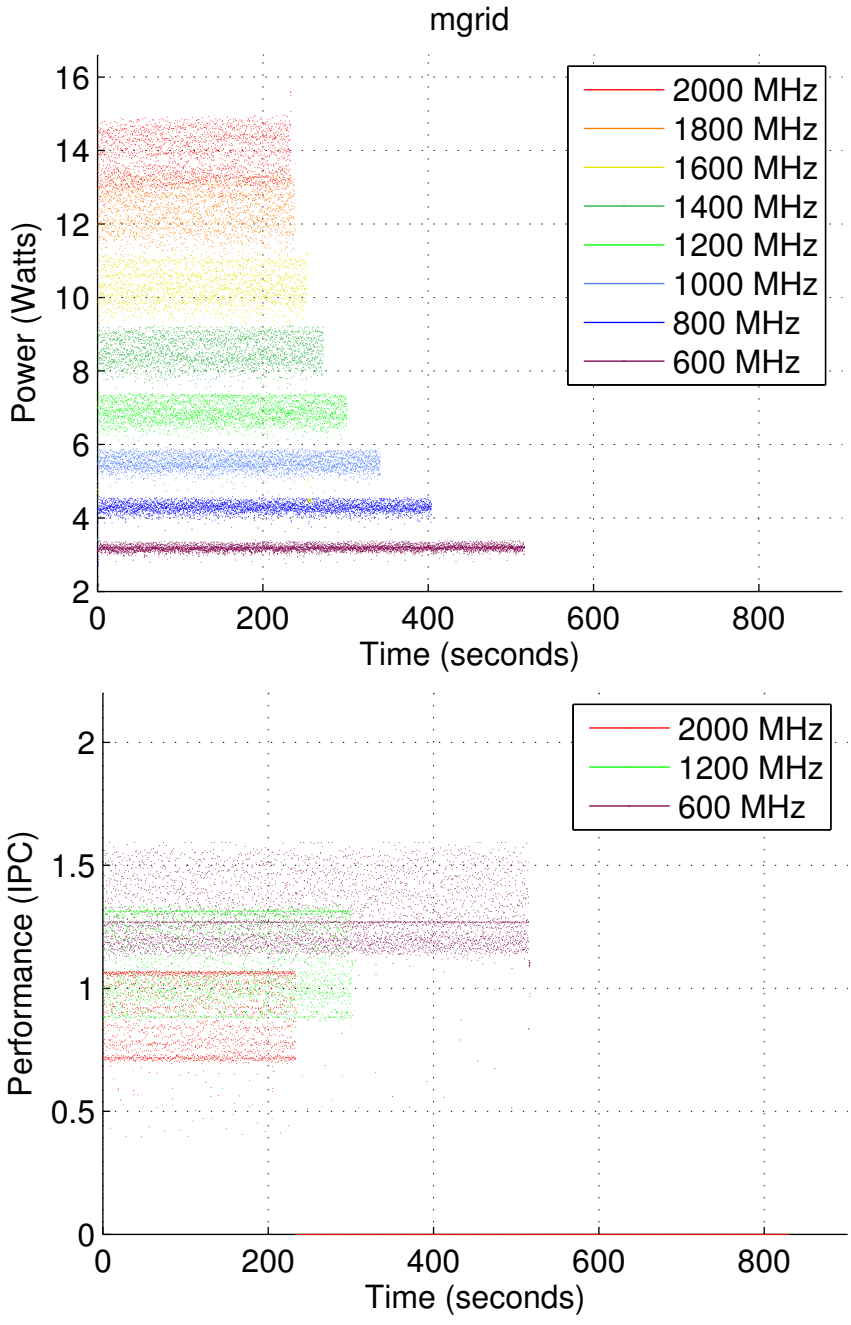


Figure B.18: Power and IPC for mgrid

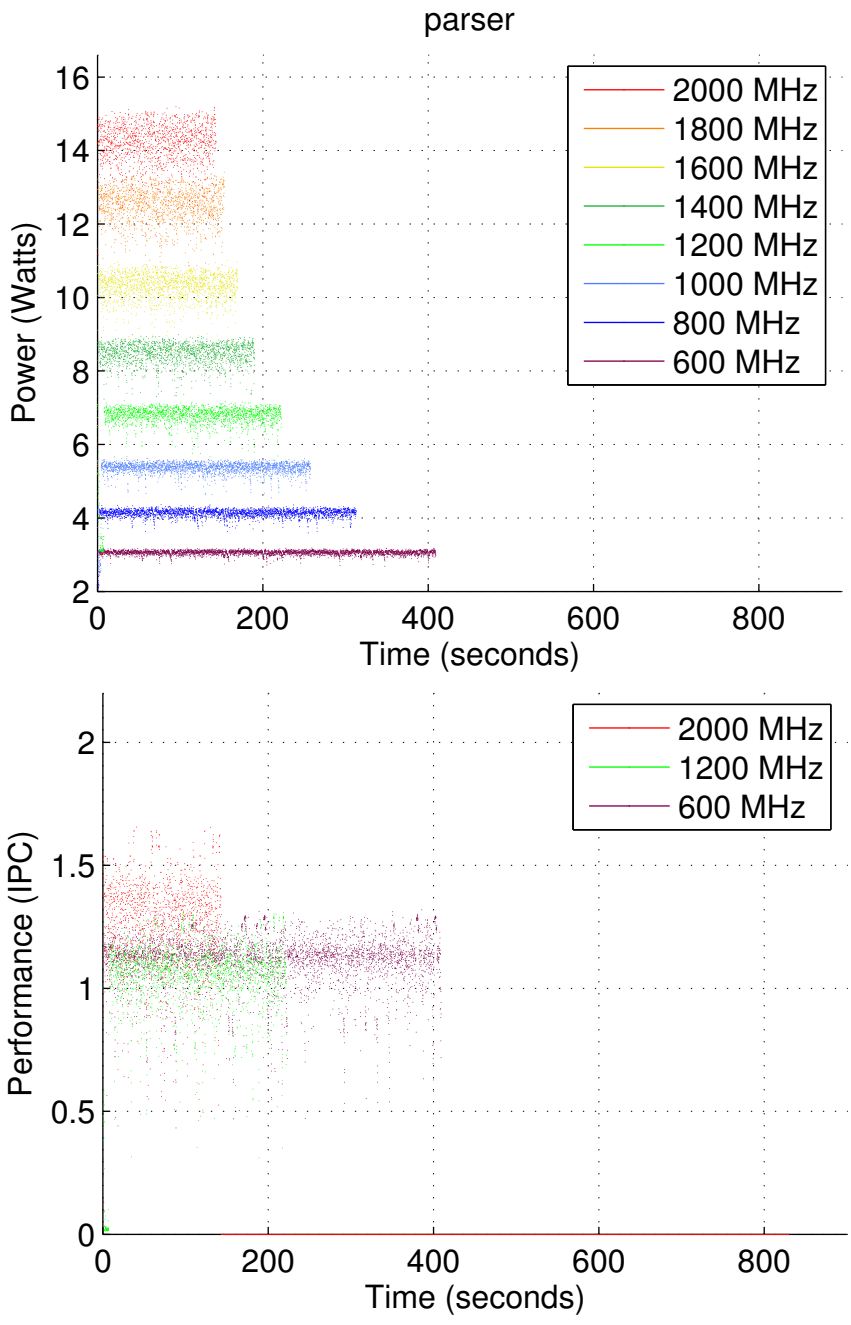


Figure B.19: Power and IPC for parser

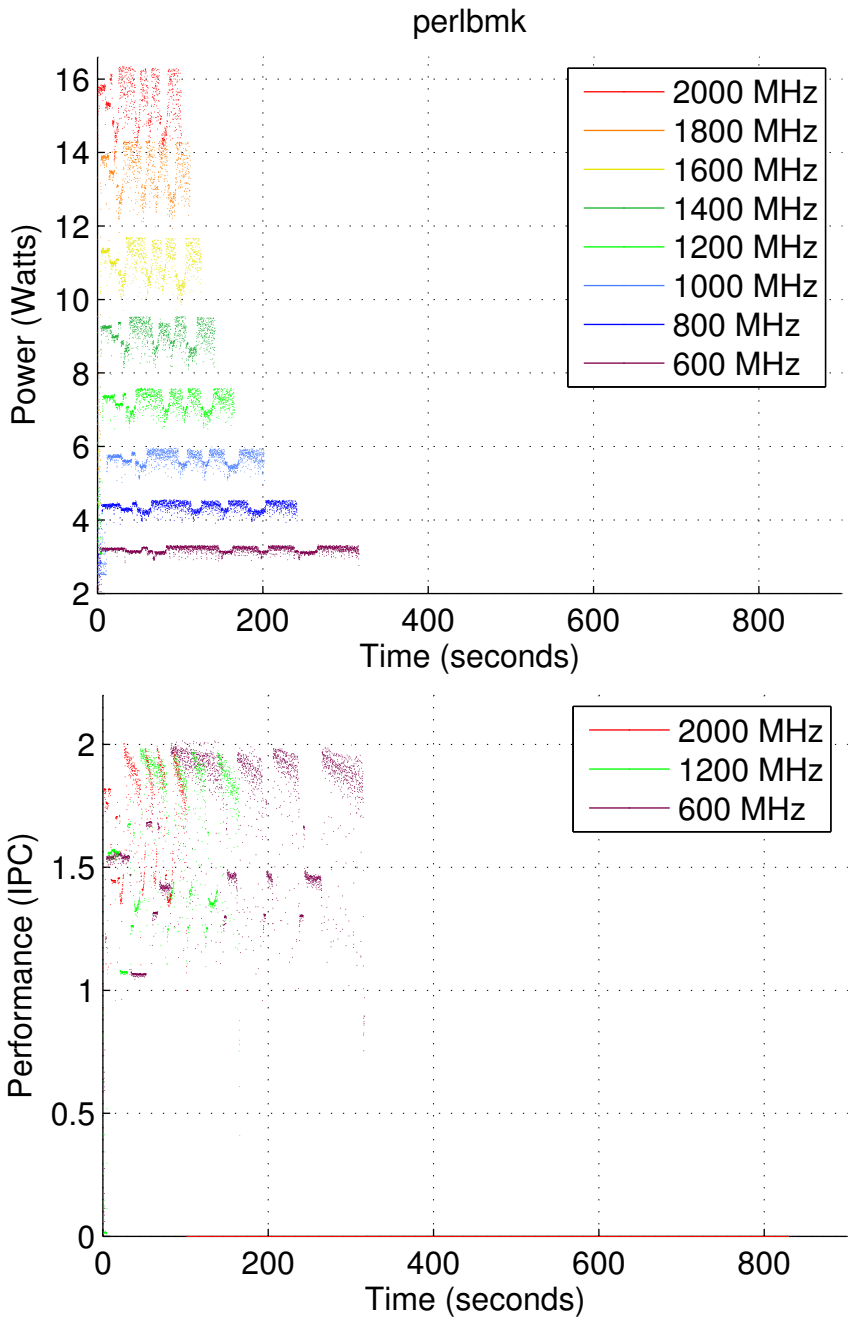


Figure B.20: Power and IPC for perlbmk

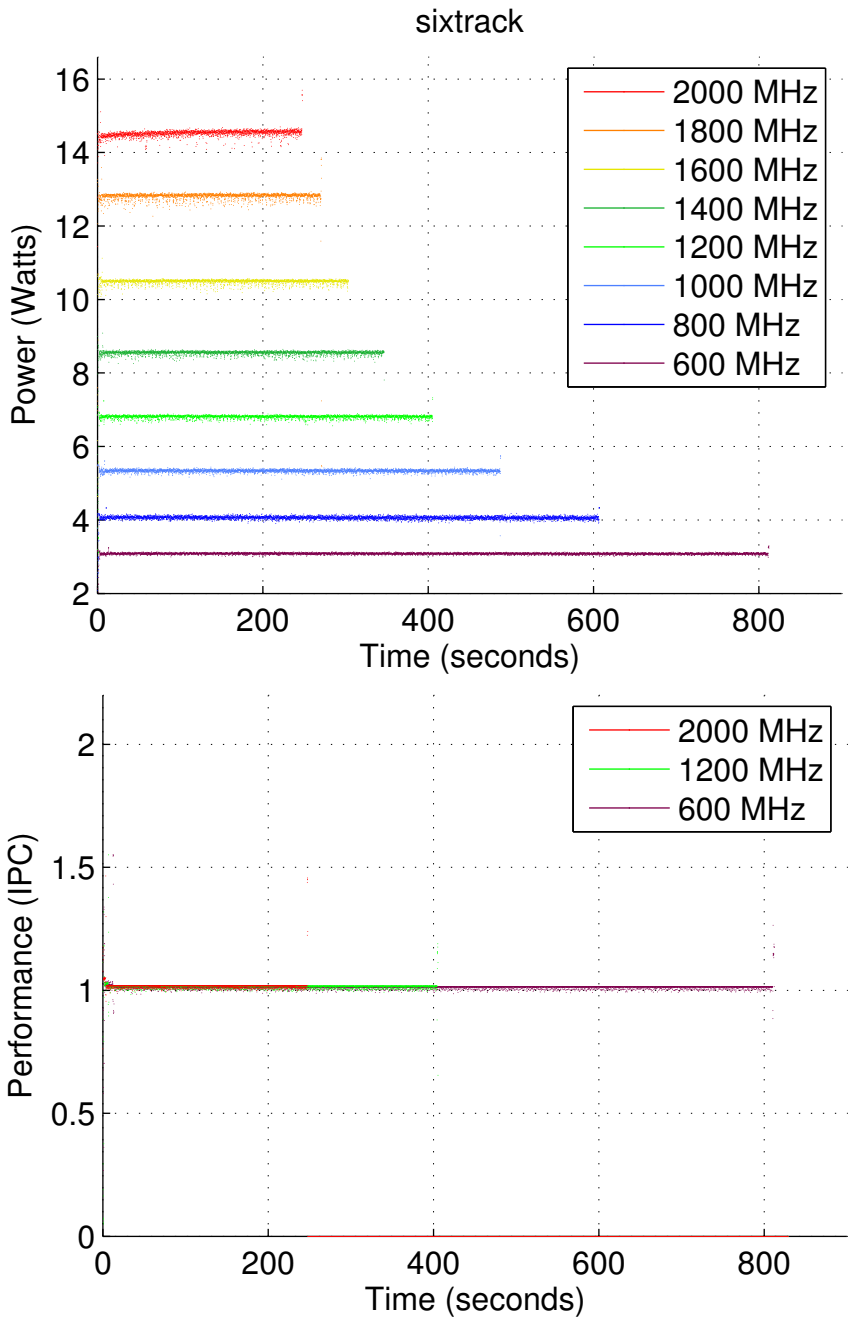


Figure B.21: Power and IPC for sixtrack

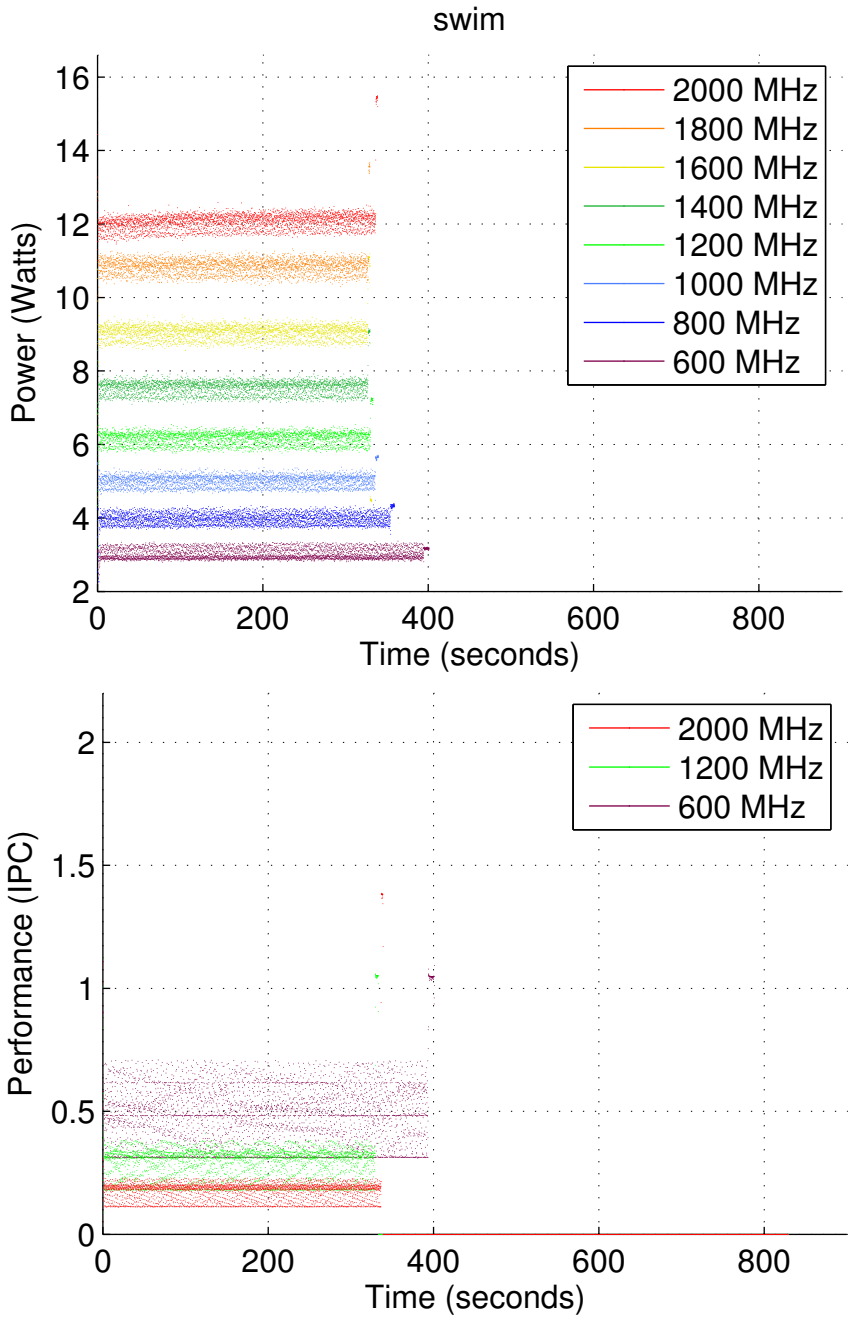


Figure B.22: Power and IPC for swim

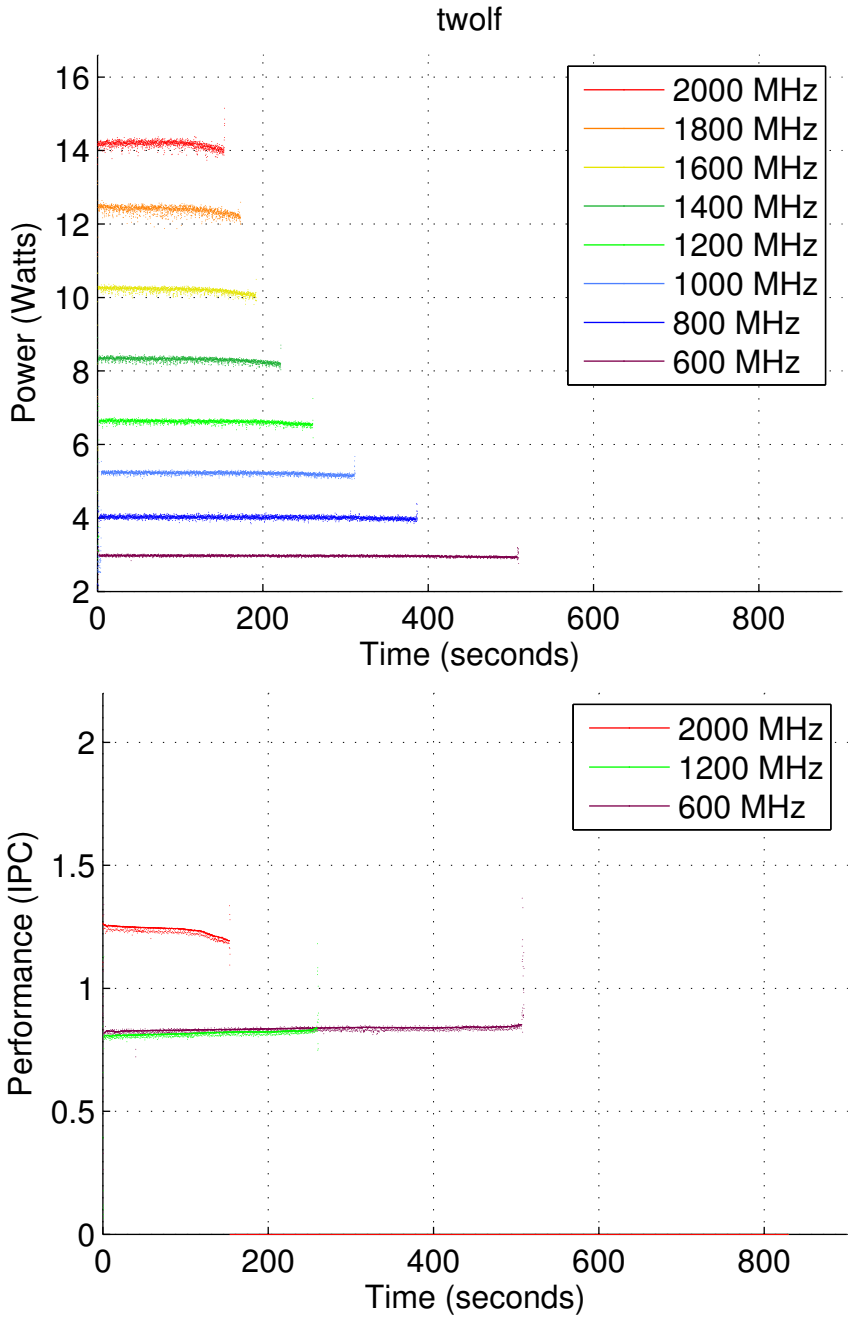


Figure B.23: Power and IPC for twolf

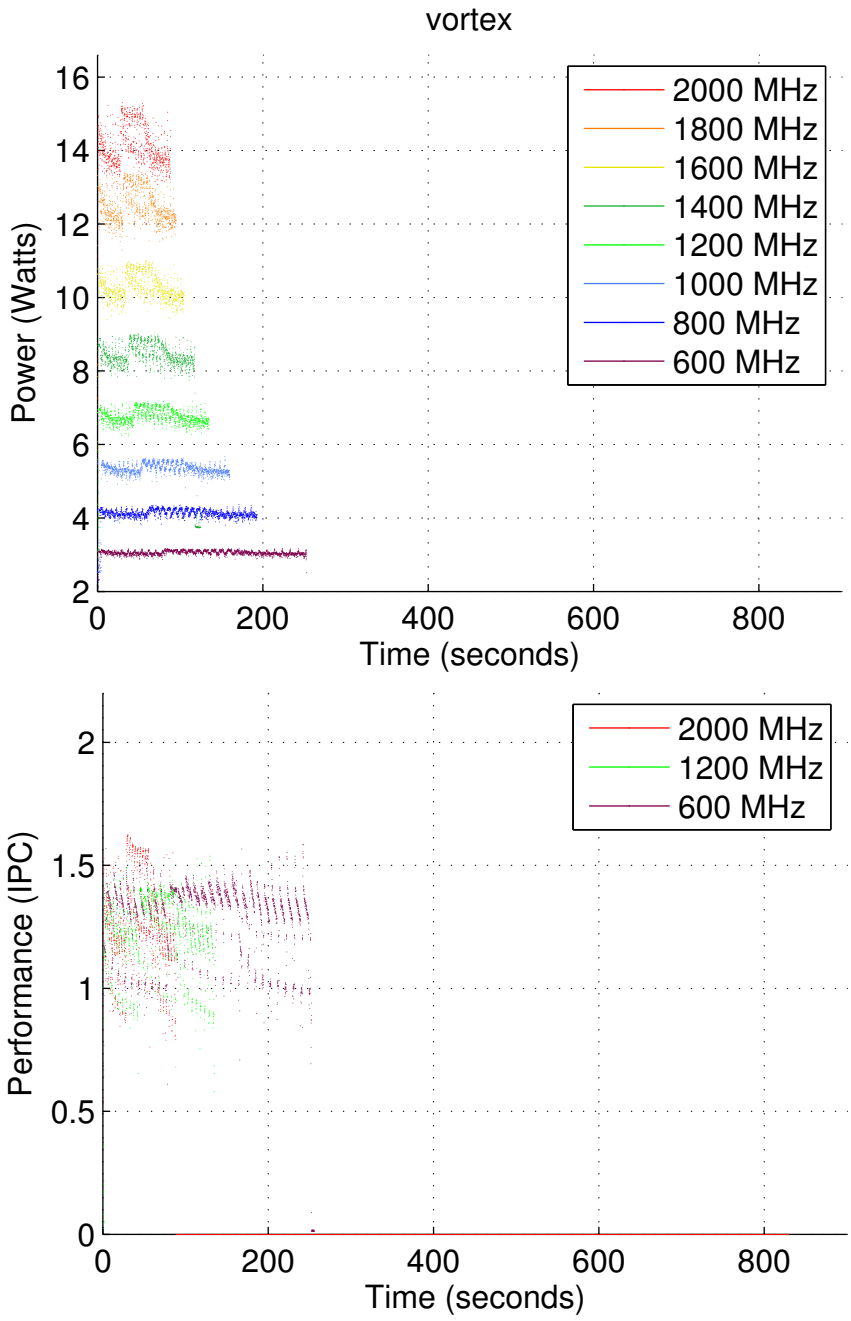


Figure B.24: Power and IPC for vortex

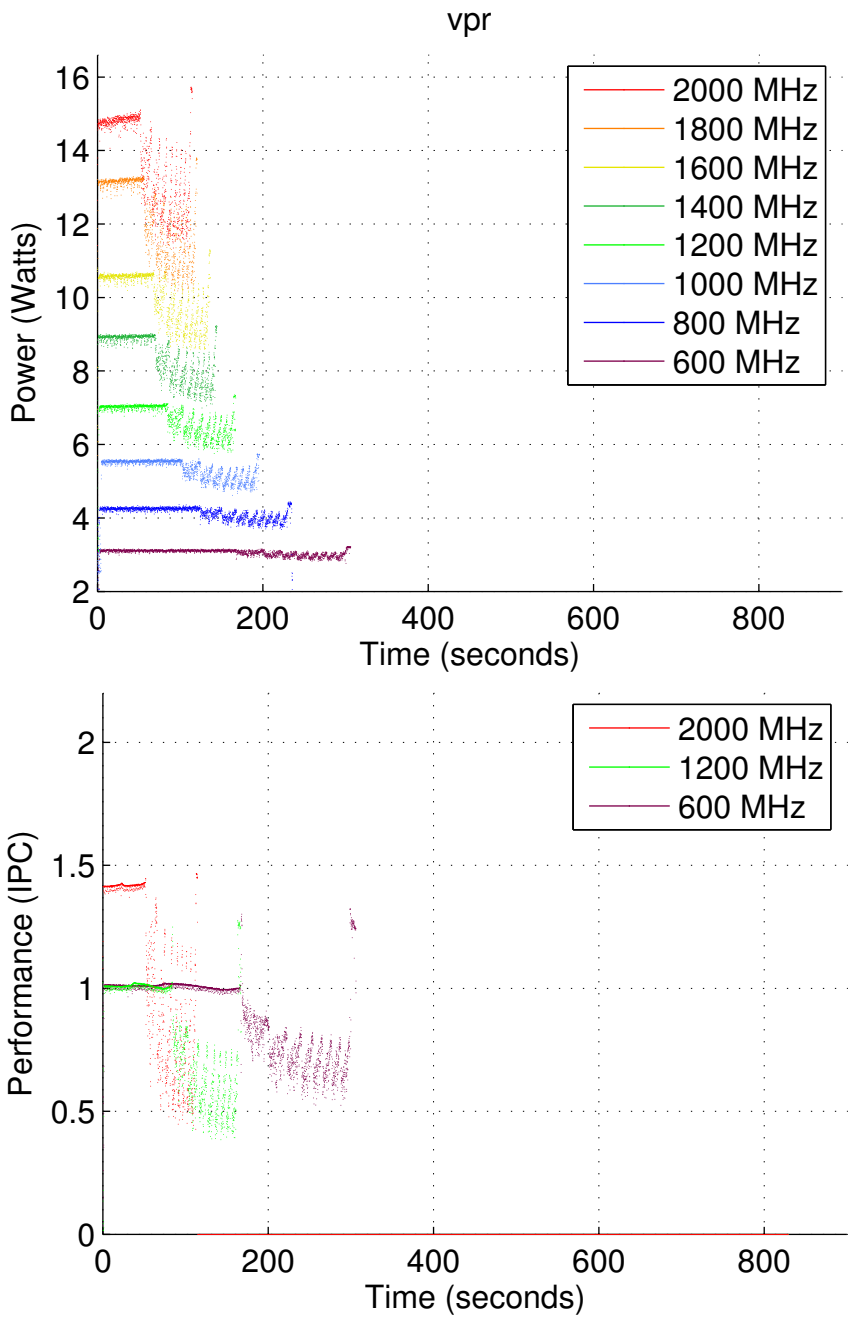


Figure B.25: Power and IPC for vpr

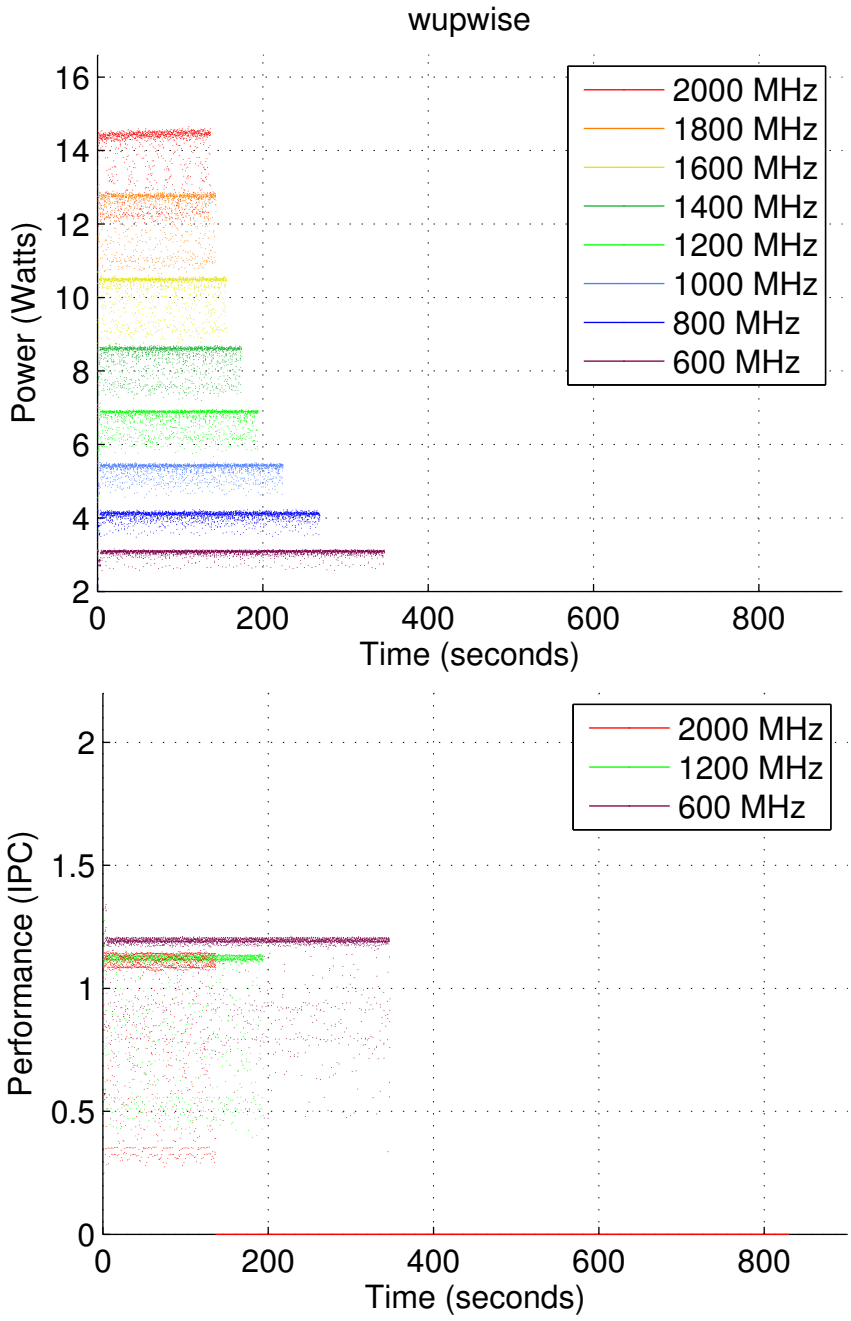


Figure B.26: Power and IPC for wupwise

Bibliography

[1] Nevine AbouGhazaleh, Daniel Moss, , Rami Melhem, and Matthew Craven. Energy management for real-time embedded applications with compiler support. In *Languges, Compilers, and Tools for Embedded Systems (LCTES)*, pages 284–293, 2003.

[2] Advanced Micro Devices. PowerNow with optimized power management. `http://www.amd.com/us-en/0,,3715_12353,00.html`, January 2006.

[3] Vikas Agarwal, M. S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock rate versus IPC: The end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 248–259, June 2000.

[4] David H. Albonesi, Rajeev Balasubramonian, Steven G. Dropsho, Sandhya Dwarkadas, Eby G. Friedman, Michael C. Huang, Volkan Kursun, Grigorios Magklis, Michael L. Scott, Greg Semeraro, Pradip Bose, Alper Buyuktosunoglu, Peter W. Cook, and Stanley E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12):49–58, 2003.

[5] C. Anderson, J. Petrovich, J. Keaty, G. Nusbaum, J.M. Tendier, C. Carter, S. Chu, J. Clabes, J. DiLullo, P. Dudley, P. Harvey, B. Krauter, J. LeBlanc, Pong-Fei Lu, B. McCredie, G. Plum, P.J. Restle, S. Runyon, M. Scheuermann,

S. Schmidt, J. Wagoner, R. Weiss, S. Weitzel, and B. Zoric. Physical design of a fourth-generation POWER ghz microprocessor. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, pages 232 – 233, 451, 2001.

[6] M. Annavaram, E. Grochowski, and J. Shen. Mitigating amdahl's law through EPI throttling. In *Proceedings of 32nd International Symposium on Computer Architecture (ISCA)*, pages 298 – 309, 2005.

[7] Frank Bellosa. The case for event-driven energy accounting. Technical Report TR-I4-01-07, Friedrich-Alexander-Universitat Erlangen-Nurnberg, 2001.

[8] William L. Bircher, Madhavi Valluri, Jason Law, and Lizy K. John. Runtime identification of microprocessor energy saving opportunitites. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 275–280, August 2005.

[9] Shekhar Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July-August 1999.

[10] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 83–94, June 2000.

[11] Kirk Cameron, Rong Ge, and Xizhou Feng. High-performance, power-aware distributed computing for scientific applications. *IEEE Computer*, 38:40–47,

November 2005.

[12] Pedro Chaparro, Jose Gonzalez, and Antonio Gonzalez. Thermal-effective clustered microarchitectures. In *Proceedings of TACS-1*, pages 80–91, June 2004.

[13] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th Annual Symposium on Computer Architecture*, pages 266–277, 2001.

[14] C. Domshlak, F. Rossi, B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: Complexity results and approximation techniques. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 215–220, 2003.

[15] A. Drake, R. Senger, H. Dun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, and M. Floyd. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2007.

[16] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36)*, pages 7–18. IEEE Computer Society, 2003.

[17] Shawn Evans. ISL6566EVAL1: Voltage regulator down solutions for intel and amd designs, January 2005. Intersil ISL6566 controller application note.

[18] Wes Felter, Karthick Rajamani, Tom Keller, and Cosmin Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th annual international conference on Supercomputing (ICS)*, pages 293–302, 2005.

[19] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. *International Symposium on Computer Architecture*, pages 148 – 157, June 2002.

[20] Kriztian Flautner and Trevor Mudge. Vertigo: Automatic performance-setting for linux. In *Operating Systems Design and Implementation (OSDI2002)*, pages 105–116, 2002.

[21] PRIMERGY BX300 Datasheet. issue date: November 16, 2004,.

[22] Soraya Ghiasi, Tom Keller, and Freeman Rawson. Scheduling for heterogeneous processors in server systems. In *Proceedings of Computing Frontiers (CF)*, pages 199–210, May 2005.

[23] Bruce A. Gieseke, Randy L. Allmon, Daniel W. Bailey, Bradley J. Benschneider, Sharon M. Britton, John D. Clouser, Harry R. Fair III, James A. Farrell, Michael K. Gowan, Christopher L. Houghton, James B. Keller, Thomas H. Lee, Daniel Leibholz, Susan C. Lowell, Mark D. Matson, Richard J. Matthew, Victor Peng, Michael D. Quinn, Donald A. Priore, Michael J. Smith, and Kathryn E.

Wilcox. A 600 MHz superscalar RISC microprocessor with out-of-order execution. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, pages 176–177, 451, February 1997.

[24] M. Gowan, L. Biro, and D. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 726–731. ACM/IEEE, June 1998.

[25] Paul Gratz, Karthikeyan Sankaralingam, Heather Hanson, Premkishore Shivakumar, Robert McDonald, Stephen W. Keckler, and Douglas C. Burger. Implementation and evaluation of a dynamically routed processor operand network. In *Proceedings of the International Symposium on Networks-on-Chips (NOCS)*, 2007.

[26] Heather Hanson. Comparison of leakage energy reduction techniques. Technical Report TR-01-18, Computer Sciences Department, University of Texas at Austin, June 2001.

[27] Heather Hanson. Sensor network for power management. unpublished research document, December 2001.

[28] Heather Hanson, M.S. Hrishikesh, Vikas Agarwal, Stephen W. Keckler, and Doug Burger. Static energy reduction techniques for microprocessor caches. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 276–283, September 2001.

[29] Heather Hanson, M.S. Hrishikesh, Vikas Agarwal, Stephen W. Keckler, and

Doug Burger. Static energy reduction techniques for microprocessor caches. *IEEE Transactions on VLSI Systems*, 11(3):303–313, June 2003.

[30] Heather Hanson and Stephen W. Keckler. Coordinated management: Power, performance, energy, and temperature. In *Proceedings of the Austin Center for Advanced Studies (ACAS) Conference*, 2005.

[31] Heather Hanson, Stephen W. Keckler, and Doug Burger. Coordinated power, energy, and temperature management for high-performance processors. In *Proceedings of the Austin Center for Advanced Studies (ACAS) Conference*, 2004.

[32] Heather Hanson, Stephen W. Keckler, Karthick Rajamani, Soraya Ghiasi, Freeman Rawson, and Juan Rubio. Power, performance, and thermal management for high-performance systems. In *Proceedings of the High Performance Power-Aware Computing Workshop*, 2007.

[33] Heather Hanson, Karthick Rajamani, Juan Rubio, Soraya Ghiasi, and Freeman Rawson. Benchmarking for power and performance. In *SPEC Benchmark Workshop*, 2007.

[34] Kim Hazelwood. Tortola: Addressing tomorrow's computing challenges through hardware/software symbiosis. In *Boston Area Architecture (BARC)*, 2006.

[35] Kim Hazelwood and David Brooks. Eliminating voltage emergencies via microarchitectural voltage control feedback and dynamic optimization. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages

326–331, 2004.

[36] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[37] Seongmoo Heo, Kenneth Barr, and Krste Asanovic. Reducing power density through activity migration. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 217–222, August 2003.

[38] ProLiant Datasheet. issue date: October 8, 2004, `http://h18000.www1.hp.com/products/quickspecs/11686_div/11686_div.pdf`.

[39] Hewlett-Packard, Intel, Microsoft Corporations, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface (ACPI) specification, revision 3.0b. `http://www.acpi.info`, October 2006.

[40] C-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[41] IBM. Managing server energy consumption using IBM PowerExecutive, 2006.

[42] IBM and MontaVista Software. Dynamic Power Management for Embedded Systems. `http://www.research.ibm.com/arl/projects/papers/DPM\_V1.1.pdf`, November 2002.

[43] 2.0 GHz Pentium M Integrated XSeries Server. `http://www-03.ibm.com/systems/i/bladecenter/ixs/4812001.html`.

[44] Intel. Pentium M processor on 90 nm process with 2-MB L2 cache datasheet, January 2005. http://www.intel.com/design/mobile/datashts/302189.htm.

[45] Intel Pentium4 Processor on 90nm Datasheet. Document Number: 300581-003; http://support.intel.com/design/Pentium4/datashts/300561.htm.

[46] Enhanced Intel SpeedStep technology. http://support.intel.com/support/processors/mobile/pm/sb/CS-007981.htm, January 2006.

[47] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, pages 93–104, 2003.

[48] Daniel A. Jimenez, Heather L. Hanson, and Calvin Lin. Boolean formula-based branch prediction for future technologies. In *Proceedings of the International Conference on Parallel Architectures and Compilation Technologies (PACT)*, pages 97–106, 2001.

[49] Philo Juang, Qiang Wu, Li-Shiuan Peh, Margaret Martonosi, and Douglas W. Clark. Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 127–130, 2005.

[50] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache-line decay: Exploiting generational behavior to reduce leakage power. In *The 28th Annual International Symposium on Computer Architecture*, pages 240–251, July 2001.

[51] Jeffrey O. Kephart, Hoi Chan, Rajarshi Das, David W. Levine, Gerald Tesauro, Freeman Rawson, and Charles Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *Proceedings of the 4th IEEE Conference on Autonomic Computing (ICAC)*, 2007.

[52] R.E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[53] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztian Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Nara ya nan. Leakage current: Moore's law meets static power. *IEEE Computer*, 36(12):68–74, December 2003.

[54] Michael Kistler, Charles Lefurgy, and Heather Hanson. Trips power sensor network. Unpublished internal research document., 2001.

[55] Ramakrishna Kotla, Soraya Ghiasi, Tom Keller, and Freeman Rawson. Scheduling processor voltage and frequency in server and cluster systems. In *Workshop on High-Performance, Power-Aware Computing (HPPAC)*, April 2005.

[56] Kouichi Kumagai, Hiroaki Iwaki, Hiroshi Yoshida, Hisamitsu Suzuki, Takashi Yamada, and Susumu Kurosawa. A novel powering-down scheme for low vt cmos circuits. In *Symposium on VLSI Circuits Digest of Technical Papers*, 1998.

[57] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commercial servers.

Computer, 36(12):39–48, 2003.

[58] Y. Li, K. Skadron, Z. Hu, , and D. Brooks. Performance, energy, and thermal considerations for smt and cmp architectures. *Proceedings of the Eleventh IEEE International Symposium on High Performance Computer Architecture (HPCA)*, February 2005.

[59] Hiroshi Makino, Yoshiki Tujihashi, Koji Nii, Chikayoshi Morishima, Yasuhi Hayakawa, Toru Shimizu, and Takahiko Arakawa. An auto-backgate-controlled MT-CMOS circuit. In *Symposium on VLSI Circuits*, pages 42–43, 1998.

[60] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA)*, pages 132–141, 1998.

[61] M. Matson, D. Bailey, S. Bell, L. Biro, S. Butler, J. Clouser, J. Farrell, M. Gowan, D. Priore, and K. Wilcox. Circuit implementation of a 600mhz superscalar RISC microprocessor. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 104–110, 1998.

[62] Rich McGowen, Christopher A. Poirier, Chris Bostak, Jim Ignowski, Mark Millican, Warren H. Parks, and Samuel Naffziger. Power and temperature control on a 90-nm itanium family processor. *IEEE Journal of Solid-State Circuits*, 41(1):229–237, January 2006.

[63] T. McPherson, R. Averill, D. Balazich, K. Barkley, S. Carey, Y. Chan, Y.H. Chan, R. Crea, A. Dansky, R. Dwyer, A. Haen, D. Hoffman, A. Jatkowski,

M. Mayo, D. Merrill, T. McNamara, G. Northrop, J. Rawlins, L. Sigal, T. Slegel, and D. Webber. 760 MHz G6 S/390 microprocessor exploiting multiple V_t and copper interconnects. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, pages 96–97, 2000.

[64] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38-8:114–117, 1965.

[65] Anil Nanduri. Dynamic Power Coordination. `http://www.intel.com/products/processor/coreduo/dynamicpowercoordination.htm`, 2006.

[66] Vijaykrishnan Narayanan. Power-aware on-chip networks. In *Austin Conference on Energy-Efficient Design (ACEED)*, 2007.

[67] Karthik Natarajan, Heather Hanson, Stephen W. Keckler, Charles R. Moore, and Doug Burger. Microprocessor pipeline energy analysis. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 282–287, 2003.

[68] Koji Nii, Hiroshi Makino, Yoshiki Tujihashi, Chikayoshi Morishima, Yasuhi Hayakawa, Hiroyuki Nunogami, Takahiko Arakawa, and Hisanori Hamano. A low power SRAM using auto-backgate-controlled MT-CMOS. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 293–298, 1998.

[69] Christopher Poirier, Richard McGowen, Christopher Bostak, and Samuel Naffziger. Power and temperature control on a 90nm Itanium-family proces-

sor. In *Proceedigns of the IEEE International Solid-States Circuits Conference (ISSCC)*, pages 304–305, March-April 2005.

[70] Michael Powell, Se-Hyun Yang, Babak Falsafi, Kaushik Roy, and T.N. Vijaykumar. Gated-V_{dd}: A circuit technique to reduce leakage in deep-submicron cache memories. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 90–95, 2000.

[71] Radisys Corporation. Endura LS855 Product Data Sheet, October 2004. `http://www.radisys.com/oemproducts/ds-page.cfm?productdatasheetsid=1158`.

[72] Karthick Rajamani, Heather Hanson, Juan Rubio, Soraya Ghiasi, and Freeman Rawson. Application-Aware Power Management. In *Proceedings of the IEEE International Symposium on Workload Characterization*, pages 39–48, October 2006.

[73] Karthick Rajamani, Heather Hanson, Juan C. Rubio, Soraya Ghiasi, and Freeman L. Rawson. Dynamic processor overclocking for improving performance of power-constrained systems. Technical report, IBM, 2005.

[74] Karthick Rajamani, Heather Hanson, Juan C. Rubio, Soraya Ghiasi, and Freeman L. Rawson. Evaluation of a dynamic processor overclocking implementation for power-constrained systems. Technical report, IBM, 2005.

[75] Karthick Rajamani, Heather Hanson, Juan C. Rubio, Soraya Ghiasi, and Freeman L. Rawson. Online power and performance estimation for dynamic power

management. Technical report, IBM, 2006.

[76] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey Chase. Ensemble level Power Management for Dense Blade Servers. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, pages 66–77. IEEE Computer Society, 2006.

[77] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson. Analysis of Thermal Monitor features of the Intel Pentium M Processor. In *Workshop on Temperature Aware Computer Systems*, June 2004.

[78] Kaushik Roy. Leakage power reduction in low-voltage CMOS designs. In *Proceedings of the International Conference on Electronics, Circuits and Systems*, pages 167–73, 1998.

[79] Hector Sanchez, Belli Kuttanna, Tim Olson, Mike Alexander, Gian Gerosa, Ross Philip, and Jose Alvarez. Thermal management system for high performance PowerPC microprocessors. In *Proceedings of the 42nd IEEE International Computer Conference (COMPCON)*, page 325, 1997.

[80] Karthikeyan Sankaralingam, Ramadass Nagarajan, Robert McDonald, Rajagopalan Desikan, Saurabh Drolia, M. S. Govindan, Paul Gratz, Divya Gulati, Heather Hanson, Changkyu Kim, Haiming Liu, Nitya Ranganathan, Simha Sethumadhavan, Sadia Sharif, Premkishore Shivakumar, Stephen W. Keckler, and Doug Burger. Distributed microarchitectural protocols in the TRIPS prototype processor. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39)*, pages 480–491, 2006.

[81] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA)*, pages 17–28. IEEE, 2002.

[82] Standard Performance Evaluation Corporation (SPEC). Spec CPU2000 benchmarks.

[83] Jayanth Srinivasan and Sarita V. Adve. Predictive dynamic thermal managment for multimedia applications. In *Proceedings of the International Conference on Supercomputing (ICS)*, pages 109–120, 2003.

[84] J.W. Tschanz, S.G. Narendra, Y. Ye, B.A. Bloechel, S. Borkar, and Vivek De. Dynamic sleep transistor and body bias for active leakage power control of microprocessors. *IEEE Journal of Solid-State Circuits*, 38(11):1838– 1845, 2003.

[85] Madhavi Valluri, Lizy John, and Heather Hanson. Exploiting compiler-generated schedules for energy savings in high-performance processors. In *International Symposium on Low-Power Electronics (ISLPED)*, pages 414–419, August 2003.

[86] Madhavi Valluri, Lizy John, and Heather Hanson. Hybrid scheduling for reduced energy consumption in high-performance processors. *IEEE Transactions on Very Large Integration (VLSI) Systems*, 14:1039–1043, 2006.

[87] Vibhore Vardhan, Daniel Grobe Sachs, Wanghong Yuan, Albert F. Harris, Sarita V. Adve, Douglas L. Jones, Robin H. Kravets, and Klara Nahrstedt.

Integrating fine-grain application adaptation with global adaption for saving energy. In *Proceedings of the 2nd International Workshop on Powe-Aware Real-Time Computing (PARC)*, 2005.

[88] Xiaorui Wang, Malcolm Ware, and Charles Lefurgy. Managing peak system-level power with feedback control. Technical Report rc2835, International Business Machines, 2005.

[89] Andreas Weissel and Frank Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, pages 238–246, October 2002.

[90] Andreas Weissel and Frank Bellosa. Dynamic thermal management for distributed systems. In *Proceedings of the First Workshop on Temperatur-Aware Computer Systems (TACS'04)*, Munich, Germany, June 2004.

[91] Kathryn Wilcox and Srilatha Manne. Alpha processors: A history of power issues and a look to the future. In *Cool Chips Tutorial: An Industrial Perspective on Low Power Processor Design*, pages 16–37, 1999.

[92] Historical weather data from weather underground, incorporated., November 2006.

[93] Fen Xie, Margaret Martonosi, and Sharad Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, June 2003.

[94] Yaakov Yaari. Event-based power estimation. Technical report, IBM, 2003.

[95] Se-Hyun Yang, Michael D. Powell, Babak Falsafi, Kaushik Roy, and T.N. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance caches. In *International Symposium on High-Performance Computer Architecture*, pages 147–157, 2001.

[96] Heng Zeng, Carla S. Ellis, and Alvin R. Lebeck. Experiences in managing energy with ecosystem. In *IEEE Pervasive Computing*, volume 4, pages 62–68, Piscataway, NJ, USA, 2005. IEEE Educational Activities Department.

[97] Huiyang Zhou, Mark C. Toburen, Eric Rotenberg, and Thomas M. Conte. Adaptive mode-control: A static-power-efficient cache design. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 61–70, 2001.

Vita

Heather Lynn Hanson was born in Iowa City, Iowa on November 10, 1969, the daughter of Dr. Gary R. Hanson and Jean A. Hanson. She received the Bachelor of Science degree in Electrical and Computer Engineering and Bachelor of Arts degree in Liberal Arts Plan II from the University of Texas at Austin in 1994, followed by a Masters of Science in Electrical and Computer Engineering in 2001.

Permanent address: 4610 Copano Court
Austin, Texas 78749

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.